2006

Data-driven fault detection using trending analysis

Min Luo
Louisiana State University and Agricultural and Mechanical College

DATA-DRIVEN FAULT DETECTION USING TRENDING ANALYSIS

A Dissertation
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

in

The Department of Electrical and Computer Engineering

by
Min Luo
B.S., Taiyuan Tech University, 1996
M.S., Tennessee Tech University, 2001
December 2006

Acknowledgements

First and foremost, I would like to express my sincere gratitude and appreciation to my advisor, Dr. Jorge Aravena, for his constant guidance to my deeper understanding of the knowledge, and his invaluable comments during the whole research work on this dissertation. They always push the limit of my ultimate research ability and never accept less than my best efforts. I also express my thanks to my minor professor Dr. Jianhua Chen, my other defense committee, Dr. Kemin Zhou, Dr. Morteza Naraghi-Pour and Dr. Bahadir Gunturk.

Special thanks are given to Heath J Leblanc for making the software more user friendly.

I would like to dedicate my work to my parents, Dr. Jinquan Luo and Dr. Shurong Li for their constant encouragement throughout my life. Thanks for always telling me I might soar even higher until I spread my wings. I want to thank my husband Yongjun for his persistent understanding and support.

Last but certainly not the least, I thank Marcie Frank in Cleveland, OH for her grammar correction of my dissertation.

Table of Contents

List of Tables

List of Figures

Abstract

The objective of this research is to develop data-driven fault detection methods which do not rely on mathematical models yet are capable of detecting process malfunctions. Instead of using mathematical models for comparing performances, the methods developed rely on extensive collection of data to establish classification schemes that detect faults in new data. The research develops two different trending approaches; one uses the normal data to define a one-class classifier. The second approach uses a data mining technique, e.g. support vector machine (SVM) to define multi class classifiers. Each classifier is trained on a set of example objects.

The one-class classification assumes that only information of one of the classes, namely the normal class, is available. The boundary between the two classes, normal and faulty, is estimated from data of the normal class only. The research assumes that the convex hull of the normal data can be used to define a boundary separating normal and faulty data.

The multi class classifier is implemented through several binary classifiers. It is assumed that data from two classes are available and the decision boundary is supported from both sides by example objects. In order to detect significant trends in the data the research implements a non-uniform quantization technique, based on Lloyd's algorithm and defines a special subsequence-based kernel. The effect of the subsequence length is examined through computer simulations and theoretical analysis.

The test bed used to collect data and implement the fault detection is a six degrees of freedom, rigid body model of a B747 100/200 and only faults in the actuators are considered. In order to thoroughly test the efficiency of the approach, the test use only

sensor data that does not include manipulated variables. Even with this handicap the approach is effective with the average of 79.5% correct detection and 16.7% missed alarm and 3.9% false alarms for six different faults.

Chapter 1

Introduction

1.1 Fault Detection and Isolation

Any complex system is liable to faults. Although good design aims to minimize the occurrence of faults, recognition that such events do occur enables system operators to respond so that the effect faults exert is minimized. Numerous applications of FDI are reported in the literature for aeronautical and aerospace systems, automotive and traffic systems, chemical processes, electrical and electronic systems, nuclear plants, power systems and transportations systems [1].

The following introduces some terms and concepts associated with fault detection and isolation. Following this general introduction, specific fault detection problems in aircraft are given. Finally, areas of interest where research work is concentrated are discussed.

In general terms a fault is any change in a system that prevents it from operating in the proper manner. Reliable detection and isolation of faults is an important part in successful maximization of productivity and safety. If a fault is to trigger the performance of some special behavior or controller reconfiguration, then some method of determining what fault has occurred is required. The procedure is called fault detection and isolation (FDI). Fault detection is a binary decision making process. Either the system is functioning properly, or there is a fault occurrence. Fault isolation is the determination of a fault source. FDI is most often considered to be a two-stage process: firstly the fact that a fault has occurred must be recognized. Secondly the type (source of the fault) should be determined so that appropriate remedial action may be initiated. If further information on

a fault is required after isolation, such as magnitude, this may be found by fault identification. Once a fault has been detected and isolated, some action is required. The nature of this response varies from triggering an alarm to notifying the operator of a fault condition which allows the operation to continue with a minimal degradation in performance. We can say that the detection and compensation of faults is one of the critical issues in the operation of high-performance systems: production equipment at power stations, chemical processes, transportation vehicles such as aircraft, space vehicles, etc.

Fault detection and isolation system schemes in plants detect and try to compensate faults in one or more of the following three subsystems: the actuator, the plant and sensors. Actuator faults are a deviation between the intended control and its realization by the actuators. Plant faults are disturbances on the plant causing a shift in the plant outputs independently of the measured inputs, and may describe plant leaks, overloads, broken down components, etc. Sensor faults are discrepancies between the measured and true values of the plant's output or input variables. The nature of faults can be classified into three categories: abrupt faults which are dramatic and persistent cause significant deviations from steady state operation, intermittent faults that are present only for very short periods of time that usually exhibit a relatively high occurrence rate after their first appearance and tend to become permanent and incipient faults that normally occur slowly over time are linked to wear and tear of components and drift in control parameters.

Fault detection and isolation systems have several major performance criteria. The design of fault detection systems demands the thorough consideration of these

criteria and tradeoffs. Based on [1], the most important criteria can be false alarm rate, missed alarm rate and time delay. There is a range of possible faults, and it is unlikely that all systems may correctly identify and respond to all fault modes. The number of false alarms that a detection system generates is very important. Response to a false alarm will tend to degrade the performance of the monitored device under no fault condition. Moreover, the missed alarm rate is more critical because no response to the real fault causes disastrous consequences. The speed at which the detection system operates is also a critical consideration. It can be termed as time delay which is the time difference between actual fault occurrence and fault detection. Ideally, remedial behavior should be triggered in a sufficiently short period so that the controlled system is never put at risk. The robustness of a fault detection scheme in the presence of modeling errors is also of major importance in most practical cases.

This research work is part of an ongoing NASA sponsored research project with the goal of improving aircraft safety. Aircraft have a long-standing tradition of being the safest among all modes of transportation. Nevertheless, as aviation continues to grow, there are concerns that unless actions are taken to drastically reduce accident rates, increased flights will lead to more accidents. Aircraft hazards can be sorted into two major categories. The first set of hazards is related to the consequences of intentional actions or failures. Usually they are addressed by aircraft security and that is out of the scope of this dissertation research. The other category refers to the undesirable consequences of unintentional actions or failures. They are addressed by the topic of aviation safety which is our research focus. These hazards caused by component failures account for roughly 25 percent of aircraft fatal accidents. The increased complexity of

aircraft will make it more difficult to identify all potential failure modes in the design phase. However, we assume that incipient hardware failures have characteristic signatures in their measurements. This is the premise of the data-driven approach to fault detection. Therefore by detecting these signatures, interpreting their significance, and alerting the flight and maintenance crew, the effect of failure can be drastically minimized.

1.2 Research Approach

Fault detection methods can be broadly classified as model-based or data-driven. Loosely speaking, model-based fault detection uses prior knowledge of the system to develop mathematical models that can, in turn, be used as references to evaluate the current data. The data-driven approach relies primarily on observations that allow the definition of a normal condition. A detailed literature review is in Chapter 2.

In model-based approaches, there exist methods that do not count on the residual for the indication of faults. One representative example is based on the use of multiple models (MM) [3]. It runs a bank of filters in parallel, each based on a model matching possible system structures due to different failures. The abrupt changes of the system are explicitly modeled by switching from one model to another in a probabilistic manner. The model probabilities are used as an indication of a failure because it provides a meaningful measure of how likely each fault mode is at a given time.

Most other model-based fault detection methods are residual-based [4][5]. In this approach, a mathematical model is created by knowing the input and the output of the system, and this model is used to compare the actual output with those nominal behaviors produced by the model and therefore residuals are formed. Then a decision needs to be

made for fault detection. Model-Based methods have been used in the production industry to overcome difficulties that arise with limit checking and claim advantages such as higher sensitivity; smaller faults can be detected and different faults can be isolated.

The obvious disadvantage of the model-based approach is the need for an accurate model of the process. When a suitable model cannot be derived from physical law, a data-driven method is applied. This method has gained increased acceptance in recent years, especially due to better techniques to define and detect patterns [6]. Unlike the model-based system, in the data-driven method an accurate mathematical model is not necessary. When the system is complex, it is difficult to develop an accurate mathematical model that represents the true system. Approximations and assumptions made in modeling compromise accuracy. Also, in model-based fault detection it is always essential to know the input to the system, which in certain cases might not be possible. Especially in an aircraft system, it is difficult to always have access to the system's input, whereas the output of the system is mostly available.

We are interested in data-driven methods that require a minimum amount of analytical information about the system. Our approach is a data-driven approach that is possible to detect faults by using the data mining technique alone based on extensive data collection of both normal and faulty performance data. In this dissertation, trending analysis is used for fault detection.

In a majority of cases, process malfunctions leave a distinct trend in the sensors monitored. These distinct trends can be suitably utilized in identifying the underlying abnormality in the process. Trending analysis can be defined as a search for patterns over time in order to identify the ways in which they change and develop, veer in new

directions or shift. Thus, a suitable classification and analysis of process trends can detect the fault earlier and lead to quick control [7]. Many papers and projects [8][9] have shown trend modeling can be used to explain the various important events happening, do malfunction diagnosis, and predict future states. The premise of fault trending analysis is that hardware failures leave characteristic signatures in the sensor data and one can use data processing tools to make visible the effect of the fault [9][11]. Instead of mathematical models that permit the determination of normal behavior, we propose to make use of an extensive collection of sensor data and create empirical descriptions of various conditions. Moreover, commercial aircrafts have extensive instrumentation, usually with built-in hardware that is redundant both physically and analytically; hence sensor faults can be neglected and measurements can still be considered reliable.

In this thesis, we develop two different trending approaches which use convex hull and one data mining technique, e.g. support vector machine (SVM). Both of them belong in the classification category that assigns a new object to one of a set of classes which are known beforehand. The classifier which performs this classification operation is based on a set of example objects. Initially, we apply the convex hull to perform a one-class classification. In one-class classification it is assumed that only information of one of the classes, which is the normal class in our research scope, is available. The boundary between the two classes, normal and faulty, has to be estimated from data of only the normal class. The convex hull can be used to define a boundary around the normal class. Next the two class classification with the aid of SVM is discussed. It is assumed that data from two classes are available and the decision boundary is supported from both sides by example objects. Moreover, close attention is paid to how to build the string-based kernel

that is more appropriate for fault detection purposes. Also, multiclass SVM is explored for fault isolation. Simulation data is from NASA's model for a B747 aircraft. The test system is a closed loop nonlinear model obtained from the software package FTLAB 747 which is simulated using Matlab Simulink. A variety of faults have been simulated to study the performance of proposed fault detection and isolation methods.

Most current trend analysis is based on univariate trends. Their trend analysis approach is based on monitoring the trend of each process variable. In summary, the univariate trend analysis has two limitations: (1) The number of false alarms is increased. (2) The dependence among process variables cannot be taken into account [12]. For the simplest case, we consider two variables. In Figure 1.1, two variables y_1 and y_2 are plotted against each other. They follow multivariate distribution. The ellipse plot represents a set of normal measurements from this distribution. The same observations are also plotted as individual charts for y_1 and y_2. From this figure, suppose there is one point indicated by the symbol ⊗ that is an abnormal measurement which is clearly outside the ellipse region. However, neither of univariate charts gives any indication of a problem for point ⊗, because it is within limits in both of the charts. The individual univariate charts effectively create a joint acceptance region shaped like a square (shown with the ellipse). This will lead to accepting a wrong measurement as good (point ⊗) but also rejecting a good measurement as bad (point ◊).

Univariate trending limitations mentioned above will worsen as the number of variables increases. In order to avoid univariate trending limitations, we provide an efficient fault detection scheme by looking at the trend of multivariable together, i.e. the

multivariate trending. From multivariate trend analysis, if we find the measurements have the tendency to leave this joint region, the measurement can be labeled as fault.

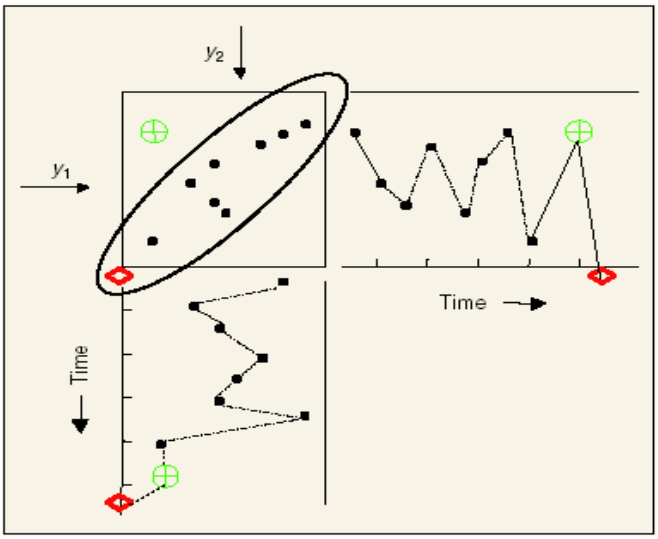


Figure 1.1 Multivariate vs. Univariate

1.3 Organization of the Dissertation

This dissertation is organized into seven chapters. In this chapter, the introduction of fault detection and the objective of this research are discussed. The literature related to fault detection methodologies are reviewed in Chapter 2. Chapter 3 discusses the one-class classification, which uses a convex hull to determine the closed boundary of a normal data class. Chapter 4 provides the details of binary classification with SVM. The focus is on how to define the string-based kernel. Fault isolation with the help of multiclass SVM is introduced in Chapter 5. In Chapter 6, we verify the proposed two approaches by testing the performance with data obtained from the B747 aircraft model. Finally, conclusions and recommendations for future work are given in Chapter 7.

Chapter 2

Literature Review

The purpose of this section is to review the common methods for fault detection. In the past decade, considerable research has been devoted to the area of system fault detection and identification (FDI). There is an abundance of literature on process fault diagnosis ranging from analytical methods to artificial intelligence and statistical approaches [13],[14]. As mentioned in Chapter 1, the fault detection approach can be either model-based or model-free.

Model-Based methods are also known as white box models. They are largely dependant on the laws of physics. The key feature is that a model is maintained that reflects the important structure and features of the system. The basic structure of many model-based detection systems is that the actual system and a model of that system are presented in parallel. Some unexpected behavior is observed in the real system that results in some symptoms being obtained. It is possible to identify possible explanations for such symptoms by comparing the predicted behavior of the model to the behavior of the faulty system. These methods have been used in the production industry to overcome difficulties that arise with limit checking and claim advantages such as higher sensitivity – smaller faults can be detected and different faults can be isolated. The obvious disadvantage is the need for an accurate model of the process. Model-free approaches rely primarily on observations (and experience) that allow the definition of a normal condition. They have gained increased acceptance in recent years, especially due to better techniques to define and detect patterns. The following subsection gives a review of model-based FDI methods. After that we focus on model-free techniques.

2.1 Model-Based Methods

The model-based fault detection can be broadly classified as qualitative or quantitative. The model is usually developed based on some fundamental understanding of the physics of the process. In quantitative models this understanding is expressed in terms of mathematical functional relationships between the inputs and outputs of the system. In contrast, in qualitative model equations these relationships are expressed in terms of qualitative knowledge about a process. Typical qualitative models are causal models and abstraction hierarchies. Details are given in the following.

2.1.1 Quantitative Model-Based Methods

Most quantitative model-based methods are residual-based. Relying on an explicit model of the monitored plant, these model-based FDI methods require two steps. The first step generates inconsistencies between the actual and expected behavior. Such inconsistencies, also called residuals, reflect the potential faults of the system. The second step chooses a decision rule for diagnosis. There exists a wide variety of residual-based approaches for linear systems, e.g. the observer-based approach, the parity space approach, and the parameter estimation approach.

There exist some model-based approaches that do not count on the residual for the indication of faults. One representative example is based on the use of multiple models (MM). It runs a bank of filters in parallel, each based on a model matching the possible system structures due to different failures. In noninteracting MM, the single-model-based filters are running in parallel without mutual interaction. Such an approach is quite effective in handling problems with an unknown structure or parameter but without structural or parametric changes. However, the problem of FDI does not fit well into such

a framework because, in general, the system structure or parameter does change as a component or subsystem fails. A notable recent advance in MM is the development of the interacting multiple-model (IMM) estimator [3]. By comparison, IMM can overcome the above weaknesses of the noninteracting MM approach by explicitly modeling the abrupt changes of the system by "switching" from one model to another in a probabilistic manner. In IMM, the model probabilities are used as an indication of a failure because it provides a meaningful measure of how likely each fault mode is at a given time.

Quantitative model-based methods have some desirable characteristics. If one has complete knowledge of all inputs and outputs of the system, including all forms of interactions with the environment, fault diagnosis would be a well-defined problem regardless of the number of faults present. On the other hand, if there is only a single sensor indicating whether the system is normal or faulty, then nothing can be diagnosed including the proper functioning of the sensor itself. The effectiveness of any diagnostic procedure is limited by the availability of sensor information [11].

A crucial need in the model-based approach is to state the significance of the observed changes with respect to the noise, unknown inputs which cannot, in any reasonable way, be modeled as random processes with known statistics. This is the general limitation of all the model-based approaches that have been developed so far. One of the popular ways of doing this is the method of disturbance decoupling. In this approach, all uncertainties are treated as disturbances and filters are designed to decouple the effect of faults and unknown inputs so that they can be differentiated [16], [17].

The other alternative is that the FDI problem has been addressed from a statistical point of view, with faults modeled as deviations in the parameter vector of a stochastic

system. Fault detection and isolation have been stated as hypotheses testing problems [18]. The key feature of this method is its ability to handle noises and uncertainties, to reject nuisance parameters and to select one among several hypotheses. First of all, FDI problems in dynamic systems are reduced to the universal static problem of monitoring the mean value of a Gaussian vector through the help of a convenient residual generation. Then different hypotheses testing methods are investigated for FDI [18].

Moreover, the types of models the analytical approaches can handle are limited to linear and some very specific nonlinear models. For a general nonlinear model, linear approximations can prove to be poor and the effectiveness of these methods might be greatly reduced. When a large-scale process is considered, the size of the bank of filters can be very large increasing the computational complexity.

2.1.2 Qualitative Model-Based Methods

The qualitative models can be developed either as qualitative causal models or abstraction hierarchies. Figure 2.1 shows the taxonomy of domain knowledge based on these two broad categories. In casual models, the cause-effect relations can be represented in the form of signed diagraphs. Causal models are a very good alternative when the quantitative models are not available but the functional dependencies are understood. Another form of model knowledge is through the development of abstraction hierarchies based on decomposition. The idea of decomposition is to be able to draw inference about the behavior of the overall system solely from the laws governing the behavior of its subsystems.

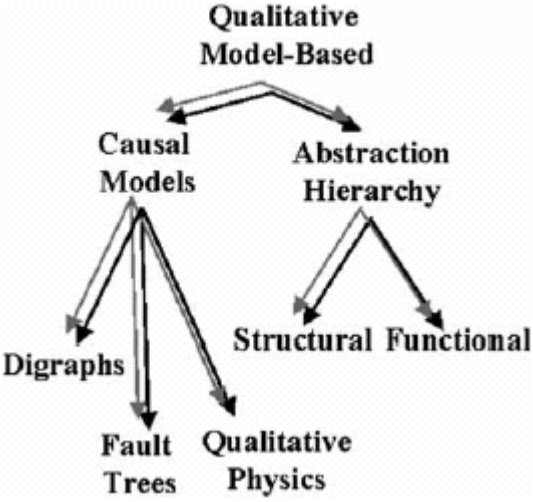


Figure 2.1 Forms of Qualitative Knowledge

Abstraction hierarchies help to quickly focus the attention of the diagnostic system to problem areas. One of the advantages of qualitative methods based on deep knowledge is that they can provide an explanation of a fault propagation path. This is indispensable when it comes to decision-support for operators. They can also guarantee completeness in that the actual fault will not be missed in the final set of faults identified. However, they suffer from the resolution problems resulting from the ambiguity in qualitative reasoning. When quantitative information is partially available, one could use the order-of-magnitude analysis or interval-calculus to improve the resolution of purely qualitative methods [11].

In the case of the qualitative model-based approaches, the combinatorial complexity is unavoidable and can only be partly alleviated with an efficient search [19]. Because of many multiple fault combinations, the search for multiple faults by specifying them explicitly as different classes and obtaining training patterns for them is not feasible.

From an industrial application viewpoint, the majority of fault diagnostic applications in process industries are based on model free or process history based approaches. This is due to the fact that process history based approaches are easy to implement, requiring very little modeling effort and a priori knowledge. Further, even for processes for which models are available, the models are usually steady-state models. It would require considerable effort to develop dynamic models specialized towards fault diagnosis applications.

2.2 Model-Free Methods

Unlike the model-based approaches where a priori knowledge about the system is needed, in model-free methods, only the availability of the large amount of historical data is needed. They are also known as the black box approach. In this research, our goal is to develop global vehicle health indicators that do not rely on mathematical models yet are capable of detecting process malfunctions. There are different ways in that data can be transformed and presented as a priori knowledge to a detection system. This is known as feature extraction. In terms of feature extraction, model-free methods can be either qualitative or quantitative in nature. Two of the major methods that extract qualitative history information are the expert systems and trend modeling methods. Methods that extract quantitative information can be non-statistical or statistical methods. Neural networks are an important class of non-statistical classifiers. Nowadays data mining is one of the most active research fields. The key advantage of data mining-based fault detection is that it can automatically generate concise and accurate detection models from large amounts of data.

2.2.1 Qualitative Feature Extraction

2.2.1.1 Expert Systems

The main advantages in the development of expert systems for diagnostic problem-solving are: ease of development, transparent reasoning, the ability to reason under uncertainty and the ability to provide explanations for the solutions provided.

There are a number of researchers who have worked on the application of expert systems for diagnostic problems. Becraft has proposed an integrated framework comprising of a neural network and an expert system [20]. A neural network is used as a first-level filter to diagnose the most commonly encountered faults in chemical process plants. Once the faults are localized within a particular process by the neural network, a deep knowledge expert system analyzes the result, and either confirms the diagnosis or else offers an alternative solution. Tirifa has proposed a hybrid system that uses signed directed graphs (SDG) and fuzzy logic [21]. The SDG model of the process is used to perform qualitative simulation to predict possible process behaviors for various faults. Those predictions are used to generate (if-then) rules that are evaluated by an expert system using information about the actual process state and fuzzy logic.

There are two types of methods for modeling knowledge for an expert system. They are shallow knowledge and deep knowledge [22]. Shallow knowledge expert systems use (if-then) type rules as the primary means of knowledge representation. These rules are formulated based on a large collection of empirical observations. In cases where the failure modes are not well known (eg: some faults are unanticipated and have very low probability of occurring), these systems are inadequate, and deep knowledge systems are more appropriate. When confronted with an unfamiliar problem an expert can resort

to “first principles”. Through an in-depth understanding of the problem, an expert can resolve problems that have not been well documented by prior observation. In this situation, the knowledge used by the expert is referred to as “deep knowledge”. This approach provides a broader knowledge base, as well as modularity for incorporating new knowledge.

However, in all applications, the limitations of an expert system approach are obvious. The expert-based fault detection system fails to generalize and detect new faults without known signatures. Knowledge-based systems developed from expert rules are very system-specific. Their representation power is quite limited, and they are difficult to update [23].

2.2.1.2 Trend Analysis

A second approach to qualitative feature extraction is the abstraction of trend information. For tasks such as diagnosis, qualitative trend representation often provides valuable information that facilitates temporal reasoning about the processes behavior. In a majority of cases, process malfunctions leave a distinct trend in the sensors monitored. These distinct trends can be suitably utilized in identifying the underlying abnormality in the process. Thus, a suitable classification and analysis of process trends can detect the fault earlier and lead to quick control. Many papers and projects have shown that trend modeling can be used to explain the various important events happening in the process, do malfunction diagnosis and predict future states. The following is an overview of some trend analysis methods and applications.

- **Triangular Episodic Presentation**

Cheung has built a formal framework for the representation of process trends [24]. A language called triangular episodic representation is formulated and used in trend extraction. It is based on temporal episodes modeled geometrically as triangles to describe the local temporal patterns in data as illustrated in Figure 2.2 and introduces triangulation to represent trends. Triangulation is a method where each segment of a trend is represented by its initial slope, its final slope (at each point, or critical point of the trend) and a line segment connecting the two critical points. A series of triangles constitutes a process trend. Through this method, the actual trend always lies within the bounding triangle which illustrates the maximum error in the representation of the trend.

As a matter of fact, this triangular episode is similar to another trend representation language, primitives. Primitives are the fundamental elements of the trend description language i.e. A(0,0), B(+,+), C(+,0), D(+,-), E(-,+), F(-,0), G(-,-) where the signs are of the first and second derivatives respectively (Figure 2.4).

- **Wavelets**

Vedam proposed a wavelet theory based nonlinear adaptive system for identification of trends from sensor data named W-ASTRA and later proposed dyadic B-Splines-based trend analysis [8]. It uses the concept of multiresolution analysis in the neural network input. Sensor data is projected onto scaling functions at different levels. First of all, the coefficients from the highest level are used to identify the primitives. If a unique primitive identification is possible then the next set of samples is collected or else the coefficients from the next lower level are used. Then W-ASTRA compare the sensor trends with their fault signature which is the segment of its trend that characterizes its behavior for a given fault class.

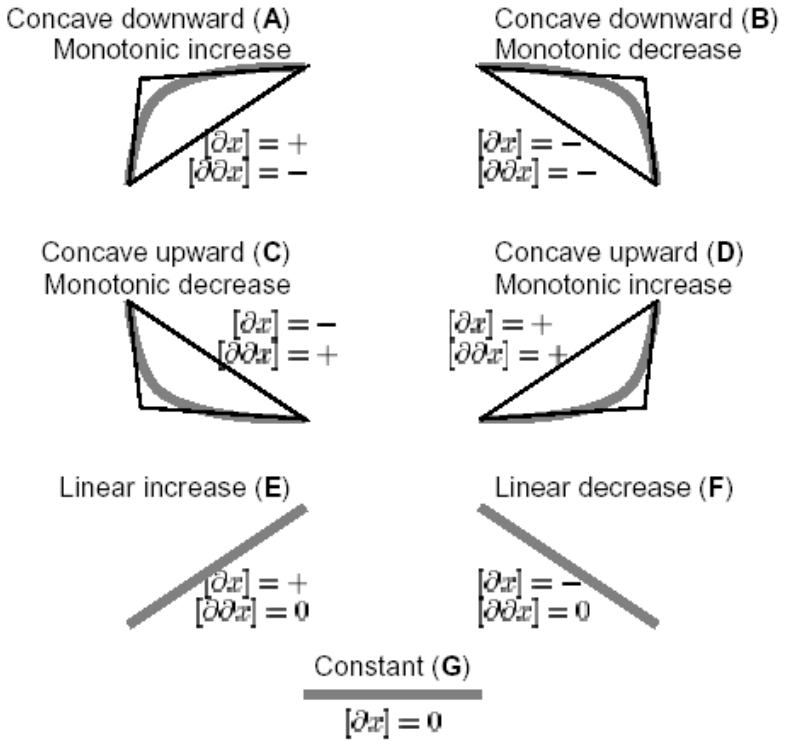


Figure 2.2 Triangular Episodic Representation

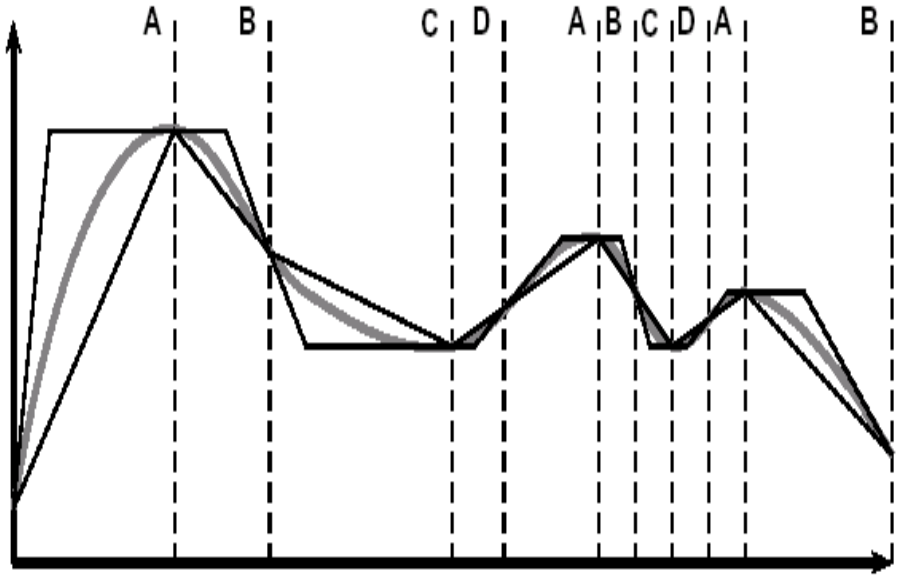


Figure 2.3 Example of Triangular Representation of Process Data

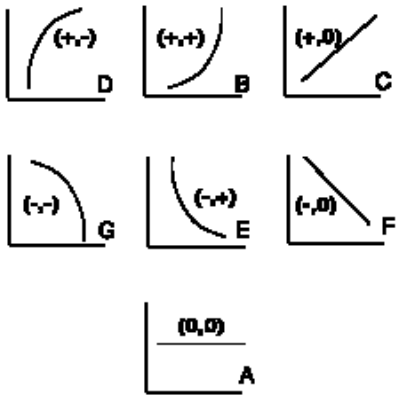


Figure 2.4 Fundamental Language: Primitive

- **Qualitative Temporal Shape Analysis**

Konstantinov proposed a generic methodology for qualitative analysis of the temporal shapes of process variables with the help of an expandable shape library that stores shapes like decreasing concavely, decreasing convexly and so on [8]. This procedure consists of three phases: analytical approximation of the process variable, its transformation into symbolic form based on the signs of the first and second derivatives of an analytical approximation function and a degree of certainty calculation.

The biggest challenge in applying trend analysis for FDI is how to automatically do trend extraction from noisy process data. In order to obtain a signal trend not too susceptible to momentary variations due to noise, some kind of filtering needs to be employed. One may simply use a filter (such as an auto-regressive filter) with a priori chosen filter coefficients (specifying the required degree of smoothing). However these types of filters suffer from the fact that they cannot distinguish well between a transient and true instability [26]. The essential qualitative characters might be distorted by these filters. Avoiding this problem requires that the trend be viewed from different time scales or different levels of abstraction. Dash proposed an interval-halving polynomial fit

approach for automatic trend extraction from noisy process data [27]. This approach parameterizes the data as a sequence of primitives with the "goodness of fit" determined with respect to noise. The interval-halving approach is a recursive method, where initially a single primitive is sought to characterize the entire data record, and when failing, the interval is halved and the process is repeated on the halved length scale until success is achieved. The procedure is recursively applied until the entire data is covered. Wavelet-based denoising is applied to remove noise. To determine the "goodness of fit", i.e., significance of error, they use the estimate of noise provided by the wavelet analysis.

2.2.2 Quantitative Feature Extraction

2.2.2.1 Neural Network

In general, the learning strategy can be classified into supervised and unsupervised learning. In supervised learning strategies, by choosing a specific topology for the neural network, the network is parameterized in the sense that the problem at hand is reduced to the estimation of the connection weights. The connection weights are learned by explicitly utilizing the mismatch between the desired and actual values to guide the search. This gives supervised techniques the ability to correctly identify a known error for which the symptoms are not known. The most popular supervised learning strategy in a neural network has been back-propagation. The neural network which utilizes the unsupervised estimation technique is known as the self-organizing neural network as the structure is adaptively determined based on the input to the network, thus unsupervised learning may be used to identify new classes of errors previously not considered. Ortega, etc. [28] constructed a neural-based diagnostic system to inspect the defects of the ropes of mining shifts automatically. A network composed of three subnetworks with error

backpropagation and momentum coefficient acquired the best results. A hierarchical neural network architecture for the detection of multiple faults was proposed by Watanabe [29]. Bakshi proposed Wavenet: a multi-resolution hierarchical neural network [30]. Wavenet is an NN with one hidden layer whose basis functions are drawn from a family of orthonormal wavelets. There are also other architectures such as self-organizing maps.

There are some limitations, however, to methods that are based solely on historic process data. It is the limitation of their generalization capability outside of the training data. This problem can be alleviated by radial and ellipsoidal units by avoiding a decision in case there are no similar training patterns in that region. This allows the network to detect unfamiliar situations arising from novel faults. Besides its lack of ability to generalize to unfamiliar regions of measurement space, networks also have difficulty with multiple faults [15]. This brings out a crucial point of distinction between model-based approaches and classifiers based on historic process data.

2.2.2.2 Data Mining -- Classification

Data mining is concerned with uncovering patterns, associations, changes, anomalies, and statistically significant structures and events in data. Simply put, it is the ability to take data and pull from it patterns or deviations which may not be seen easily to the naked eye. Another term sometimes used is knowledge discovery.

The recent rapid development in data mining has made available a wide variety of algorithms drawn from the fields of statistics, pattern recognition, machine learning, and database. The key advantage of data mining-based fault detection is that it can automatically generate concise and accurate detection models from large amounts of data.

The methodology itself is general, and therefore can be used to build fault detection systems for a wide variety of computing environments.

Data mining techniques such as Support Vector Machines (SVM) and the Association Rule have been investigated in the context of fault detection. SVM is a relatively new type of learning algorithm. When used for classification, SVM separates a given set of binary-labeled training data with a hyperplane that is maximally distant from them (known as maximal margin hyperplane) [31]. For cases in which no linear separation is possible, they can nonlinearly map the input vector into a high dimensional feature space where the data can be linearly classified. The hyperplane found by the SVM in feature space corresponds to a nonlinear decision boundary in the input space. Given a test instance, its distance from the hyperplane can be calculated and, following some threshold, it can be determined if the instance is anomalous. Sample applications in detecting novel data can be found in [32][33]. However, as a classifier, prior knowledge for the learned domain and novel region is needed to provide a learning basis for SVM tools.

There has been an increased interest in data mining-based approaches to build detection models for intrusion detection systems (IDS). These models generalize from both known attacks and normal behavior in order to detect unknown attacks. They can also be generated in a quicker and more automated method than manually encoded models that require difficult analysis of audit data by domain experts. Several effective data mining techniques for detecting intrusions have been developed [34][35][36], many of which perform close to or better than systems engineered by domain experts. In [37], the idea is to first compute the association rules and frequent episodes from audit data

which capture the intra- and inter- audit record patterns. These patterns are then utilized, with user participation, to guide the data gathering and feature selection processes.

In some cases, all positive examples are alike but each negative example is negative in its own way. Negative examples come from an unknown number of negative classes. In other cases, one class is sampled very well, while the other class is severely undersampled. The measurements on the undersampled class might be very expensive or difficult to obtain. The objective becomes making a description of a target set of objects and to detect which new objects resemble this training set. The difference with conventional classification is that in one-class classification only examples of one class are available. The objects from this class are called the target objects. All other objects are named the outlier objects. In the literature, a large number of different terms have been used for this problem. The term one-class classification originates from [38], but also outlier detection and novelty detection [39] are used. One possible approach to one-class classification is to use a density method which directly estimates the density of the target objects. By assuming a uniform outlier distribution and by the application of Bayes' rule, the description of the target class is obtained. For instance, in [40] the density is estimated by a Parzen density estimator. In [39] not only the target density is estimated, but also the outlier density. Unfortunately, this procedure requires a complete density estimate in the complete feature space. Especially in high dimensional feature space this requires huge amounts of data. Furthermore, it assumes that the training data is a typical sample from the true data distribution. In most cases the user has to generate or measure training data and one might not know beforehand what the true distribution might be. This makes the application of the density methods problematic.

Alternatively, boundary methods have been developed which only focus on the boundary of the data. They try to avoid the estimation of the complete density of the data and therefore work with an uncharacteristic training data set. For the boundary methods, it is sufficient that the user can indicate just the boundary of the target class by using examples. An attempt to train just the boundaries of a data set is made in [41]. Neural networks are trained with extra constraints to give closed boundaries. In [42], a new type of one-class classifier is presented, the support vector data description. It models the boundary of the target data by a hypersphere with minimal volume around the data. The boundary is described by a few training objects, the support vectors.

We develop one-class classification with the convex hull concept and binary classification (normal and faulty) with SVM. Using the normal data collected we use a standard algorithm to define its convex hull. Measurements that fall outside the convex set are classified as indicating a fault. When both normal and faulty data are available, we consider using SVM for binary classification due to the excellent generalization performance (accuracy on test data) in practice.

Chapter 3

Fault Trending Analysis Part I – Convex Hull

3.1 One-Class Classification

Much effort has been expended to solve the fault detection problem with classification techniques. Although the problem of classification is far from solved in practice, the one-class classification is also of interest. In the fault detection problem, all positive examples are alike but each negative example is negative in its own way. For instance, the different faults considered in this thesis are the failure in each of the four control surfaces: elevator, aileron, rudder, stabilizer and in the engine. Not only are these five different faults different from the normal data, they quite differ from each other. Therefore, all the faulty data comes from a variety of faulty classes. In other cases, one-class is sampled very well, while the other class is severely undersampled. The measurements on the undersampled class might be very expensive or difficult to obtain. For example, in a machine monitoring system where the current condition of a system is examined, an alarm is raised when the machine has a problem. Measurements on the normal working condition of a machine are very inexpensive and easy to obtain. On the other hand, measurements of faults would require the destruction of the machine in all possible ways. Therefore, it is rather expensive, if not impossible, to generate all faulty situations [43].

One possible solution is one-class classification [38]. The goal in one-class classification is to make a description of a target set of objects and to detect whether new objects resemble this training set. If yes, new objects belong to the target class; otherwise, new objects are in the outlier class. In Chapter 2, we already introduced several different one-class classification methods. Here we are more interested in the boundary approach

that estimates the boundary of the target class, i.e. the normal class with the normal data available only.

The most straightforward method to obtain a one-class classifier is to estimate the density of the training data and to set a threshold on this density [40]. Several distributions can be assumed, such as Gaussian or a Poisson distribution. When the sample size is sufficiently high and a flexible density model is used, this approach works very well. Unfortunately, this method requires a complete density estimate in the complete feature space. Especially in high dimensional feature space this method requires huge amounts of data. Furthermore, it assumes that the training data is a typical sample from the true data distribution. In most cases the user might not know beforehand what the true distribution might be. All these disadvantages make the application of the density methods problematic [43].

By comparison, boundary methods only focus on the boundary of the targeted data. They avoid estimating the complete density of the data and therefore working with an uncharacteristic training data set. This not only gives an advantage when just a limited sample is available, it is even possible to learn from data when the exact target density distribution is unknown [38]. Although in principle the boundary methods are more efficient than the density estimation, it is not directly clear how one should define a boundary around a target set, how to define the resemblance of an object to a target set and where to put the threshold. In most cases a distance to the target set is defined as a function of Euclidean distances between test objects, between the test object and target objects, and between the target objects themselves. However, this requires well-defined distances in the feature space and thus well-scaled features.

The first boundary method is the K-center method that covers the dataset with k small balls with equal radii [45]. The ball centers μ_k are placed on training objects so that the maximum distance of all minimum distances between training objects and the centers is minimized. In the fitting of the method to the training data, the following error is minimized:

$$\varepsilon_{k-center} = \max_i (\min_k \|x_i - \mu_k\|^2) \tag{3.1}$$

The K-centers method uses a forward search strategy starting from a random initialization. The radius is determined by the maximum distance to the objects that the corresponding ball should capture. By this construction the method is sensitive to the outliers in the training set, but it will work well when clear clusters are present in the data. When the centers have been trained, the distance from a test object z to the target set can be calculated. This distance is now defined as:

$$d_{k-center}(z) = \min_k \|z - \mu_k\|^2 \tag{3.2}$$

The other representative methodology is spherical data description [42]. A model that gives a closed boundary, a hypersphere around the data is defined. The sphere is characterized by a center a and radius R in Figure 3.1 and it demands that the sphere contains all training objects.

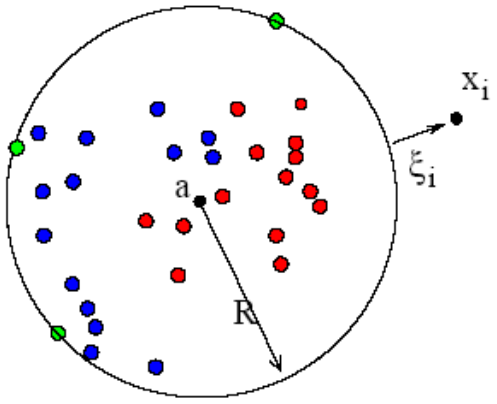


Figure 3.1 The hypersphere containing the target data, described by the center a and radius R

They start by defining the error to minimize

$$\varepsilon(R,a) = R^2 + C\sum_i \xi_i \tag{3.3}$$

with the constrain so that all target objects lie within the hypersphere:

$$\|x_i - a\|^2 \le R^2 + \xi_i,\ \xi_i \ge 0 \tag{3.4}$$

Because they can give an expression for the center of the hypersphere a, it is able to test if a new object z is accepted by the description. A test object z is accepted when the distance to the center is smaller or equal to the radius R.

Although the volume is not always actively minimized in the boundary methods, most methods have a strong bias towards a minimal volume solution. How small the volume is depends on the fit to the data. Because the boundary methods heavily rely on the distance between objects, they tend to be sensitive to scaling of the features. On the other hand, the number of objects that is required is smaller than one in the case of density methods.

For data-driven fault detection it is essential to be able to differentiate between sensor data generated by normal operations from data arising from faulty conditions and we need to do this differentiation without relying on detailed mathematical models. A simple approach for the characterization of normal behavior would be to use historical data to compute normal ranges for each measured variable, i.e. static "red-line" limits. For example, a fault in a heat regulator might be detected when a particular temperature gets higher than a given threshold. Such limits are popular because they are relatively easy to specify and use [46]. One could use the criterion that if all sensors reading are within normal ranges then the situation is normal. This conventional approach is widely used in the process industry where alarm panels provide visible indication when variables go out of range.

But they have numerous weaknesses, which are becoming increasingly significant as we move toward fault detection for aircraft including:

1) Late or missed alarms --- red-lines are relatively weak (wide) bounds, detecting faults only once they become critical and often dangerous. Earlier detection would support a wider range of recovery procedures, including preventative maintenance that would extend mission life.
2) False alarms --- red-lines are traditionally made quite wide intentionally, in large part to avoid false alarms. Nevertheless, such false alarms still occur routinely, sometimes resulting in operators eventually ignoring red-line alarms in those troublesome sensors altogether.
3) Failure to track system changes --- predefined red-lines fail to capture changes during the gradual degradation of components.

Especially for aircrafts the “red-line” limits approach may not be completely safe because simultaneous extreme values in some variables may lead to an unrecoverable condition.

It is our contention that for aircrafts this approach may not be completely safe as simultaneous extreme values in some variables may lead to an unrecoverable condition. In Figure 3.2 we show a “normal” variation of pitch and roll angles for a B747. The variations have been obtained using a simulator and introducing zero-mean random fluctuations to all actuators.

We note that requiring each variable to be between given bounds is equivalent to assuming the normal set to be a hypercube (under suitable normalization). It is known any linear constraint defines a convex set and a set of N simultaneous linear constraints defines the intersection of N convex set, which is also a convex set. Therefore, we can think normal data stay in a convex set. In the case of commercial flights, most of the time an aircraft operates in the neighborhood of a trimming point, equivalent to the operating point in an industrial process. Hence, small variations can be well described by a linear model. Motivated by this heuristic justification, in this work we consider the assumption that normal operation sensor data should cluster in a convex set centered at the trimming values. The approach proposed here is to find a suitable enclosure for the conditions known to be normal, instead of a rigid hypersphere in [42]. To minimize the chance of accepting faulty data, the volume of the normal convex data set has to be minimized. In the next section, this minimization is achieved by the description of the convex hull which is the smallest convex set.

How that set is determined and the effect on missed and false alarms are issues discussed here. The size of this normal convex set is critical and there is a clear tradeoff between its size and missed/false alarm rates. For size zero everything is abnormal so we have 0% missed alarms and 100% false alarms. If the normal convex set is extremely large, then everything is normal and one has 100% missed alarms and 0% false alarms. We also consider the curse of dimensionality as it applies to this case; testing if a point is in the inside of a high dimensional convex set can be computationally demanding hence one must seek ways to reduce the number of variables that are being considered.

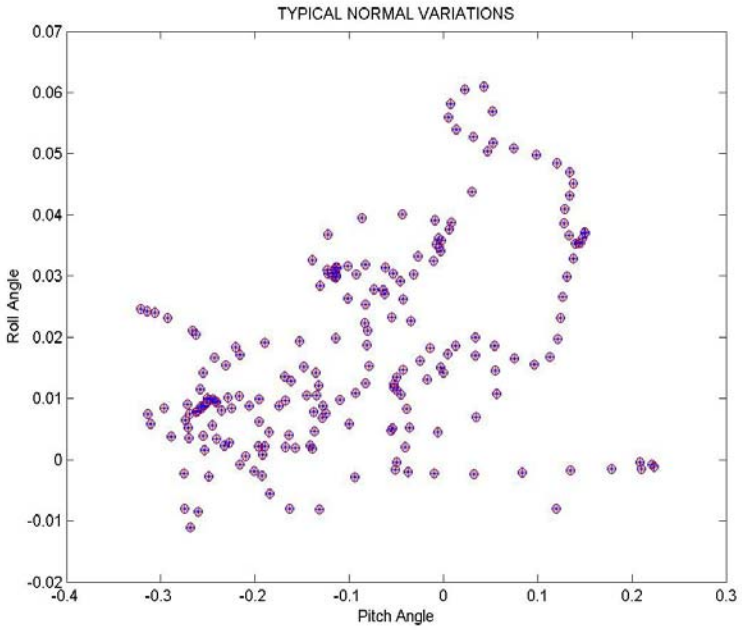


Figure 3.2 Pitch Angle vs. Roll Angle under Normal Flying Conditions

3.2 Convex Hull Classification

Since one always has only a finite number of data points, formal determination of a convex set of normal values cannot be done through experimental data collection. At any given time, we use instead the convex hull of the points "known to be normal."

Therefore, it is necessary to also consider the possibility of modifying the set if additional normal data is received; i.e., one must include the option of recursively defining the convex hull. We first discuss the classification using convex hull and its numerical complexity. Then we discuss the method proposed to modify the hull if new data, known to be normal, falls outside the current hull.

3.2.1 Convex Hull Algorithm

A set is convex if every line segment connecting two points in the set is fully contained in the set (Figure 3.3).

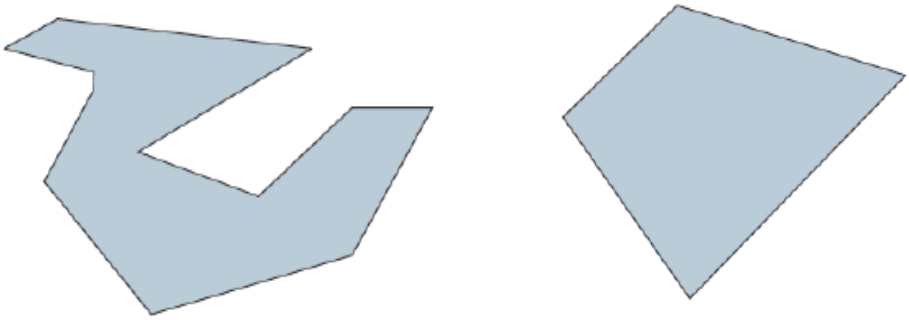


Figure 3.3 Non-convex Set Vs Convex Set

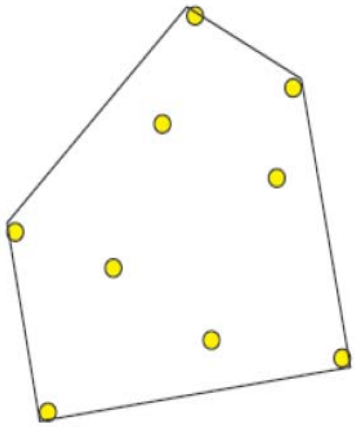


Figure 3.4 Convex Hull

The convex hull of a set of points is the smallest convex set that contains the points (Figure 3.4).

In this thesis, we use the Quickhull algorithm [47] to construct the normal convex set. The convex hull is represented by a set of facets and a set of adjacency lists giving the neighbors and vertices for each facet. The boundary elements of a facet are called ridges. Each ridge signifies the adjacency of two facets.

In Quickhull algorithm, it is assumed that the normal input points are in general position (i.e, no set of $d+1$ points define a $(d-1)$ flat), so that their convex hull is a simplicial complex. A simplicial complex is a space with a triangulation. Formally, a simplicial complex in R^n is a collection of simplicies in R^n. A simplex, sometimes called a hypertetrahedron, is the generalization of a tetrahedral region of space to n dimensions. In one dimension, the simplex is the line segment. In two dimensions, the simplex is the convex hull of the equilateral triangle. In three dimensions, the simplex is the convex hull of the tetrahedron. Figure 3.5 shows the graphs for the n-simplexes with $n = 2$ to 7.

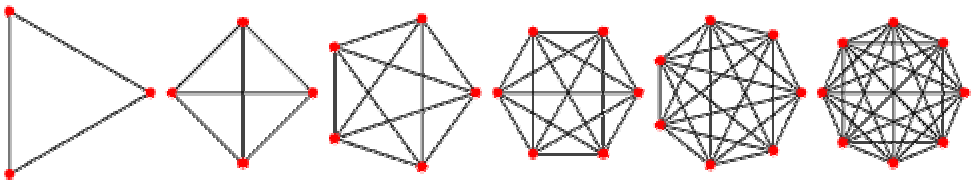


Figure 3.5 Simplex in n dimension

The Quickhull algorithm can be extended to singular data by triangulating non-simplicial facets [47][48].

We represent a d-dimensional convex hull by its vertices and (d-1)-dimensional faces. Each facet includes a set of vertices, a set of neighboring facets, and a hyperplane equation. The $(d-2)$-dimensional faces are the ridges of the convex hull. Each ridge is the intersection of the vertices of two neighboring facets. For notational purposes, we let $P_r := \{p_1,...,p_r\}$ be the set of r measurements and $CH(P_r)$ its convex hull. In the end, Quickhull returns the indices of the points (vertices) in P_r that comprise the facets of the convex hull of P_r.

In d-dimensions, a (d-1)-dimensional facet is a hyperplane. If the distance of a point to the hyperplane is positive, the point is above the hyperplane. A hyperplane can be represented by its unit normal and its offset from the origin. The Hessian normal form of the plane is

$$\hat{n} \cdot p = -s \tag{3.5}$$

$\hat{n} = \{n_x, n_y, n_z, ...\}$ is the unit norm vector. In N-dimensional geometry, the normal $\hat{n}$ at the point x_0 is a generalized cross product of $(N-1)$ edge vectors.

$$\hat{n} = \begin{vmatrix} \hat{x} & \hat{y} & \hat{z} & ... & \hat{w} \\ (x_1 - x_0) & (y_1 - y_0) & (z_1 - z_0) & ... & (w_1 - w_0) \\ (x_2 - x_0) & (y_2 - y_0) & (z_2 - z_0) & ... & (w_2 - w_0) \\ ... & ... & ... & ... & ... \\ (x_{N-1} - x_x) & (y_{N-1} - y_0) & (z_{N-1} - z_0) & ... & (w_{N-1} - w_0) \end{vmatrix}$$

As usual, we can form the normalized normal $\hat{n} = \frac{\hat{n}}{\|\hat{n}\|}$.

$(\hat{x}_k - \hat{x}_0)$ are $(N-1)$ edge vectors tangent to the normal vector at a point $\hat{x}_0$.

s is the offset from the origin.

The distance to a point p_r is

$$D = \hat{n} \cdot p_r + s \tag{3.6}$$

If $D > 0$, p_r it is in the same direction of $\hat{n}$, otherwise, it is the opposite direction of $\hat{n}$.

After the value of unit normal vector $\hat{n}$ is determined by the cross product of edge vectors, we need to decide its direction. Since the convex hull is determined, we can use any point p_0 known to be inside the hull to determine the direction of the outward-pointing unit normal vector $\hat{n}$. The distance from p_0 to the hyperplane is calculated. If the distance is positive, the normal vector $\hat{n}$ of the hyperplane is in same direction of p_0, which points to the inside of convex hull. If the distance is negative, the normal vector $\hat{n}$ of the hyperplane is in the opposite direction of p_0, which points to the outside of the convex hull.

The fault detection decision is based on the following principle:

Principle: If the point p_r is below all facets, it is inside the convex hull. Otherwise, the point p_r is outside $CH(P_{r-1})$ [47].

We can use the sign of distance of p_r to all facets as described above to determine if the p_r is inside or outside the convex hull.

The complete decision process is as follows.

Procedure:

1) Calculate the signed distance $D_0 = \hat{n}_0 \cdot p_0 + s_0$ for any point p_0 which is inside $CH(P_{r-1})$

2) If $D_0 > 0$, then $\hat{n} = -\hat{n}_0$, $s = -s_0$

Else if $D_0 < 0$, then $\hat{n} = \hat{n}_0$, $s = s_0$

Once the direction of the normal vector is defined, we can calculate the signed distance $D_i = \hat{n}_i \cdot p_r + s_i$ for every facet of $CH(P_{r-1})$. If any $D_i > 0$, the point p_r is in $CH(P_{r-1})$ outside set, i.e., p_r is a faulty point.

3.2.1.1 Convex Hull Computation Complexity

Theorem: [47] if an execution of Quickhull is balanced, its expected complexity is $O(n \log r)$ for $d \leq 3$ and $O(nf_r / r + f_r)$ for $d \geq 4$.

d is the dimension, n is the number of input points, r is the number of vertices, and f_r is the maximum number of facets of r vertices. Moreover, it is known that f_r increases with d. For example, when $d = 4$, $f_r = 101$; $d = 5$, $f_r = 216$. In the convex hull classification we need to test every facet for a new coming point, so the large number of f_r will increase the calculation effort.

We already estimated the complexity of calculating a convex set as the number of measurements increases. For the case of the B747, the number of measurements can easily exceed 70 and even concentrating on the basic 12 state variables may make trending computationally intense. Therefore, one necessary step before applying the convex hull classification is to reduce dimensionality of the measurements.

3.2.1.2 Recursively Increasing Convex Hull

If a new point is known to be normal and it is outside the convex hull, the convex hull should be able to extend itself to include this new point, to transform from $CH(P_{r-1})$ to

$CH(P_r)$. In order to add a point to a convex hull, we have to first identify the facets below the point. These are the visible facets for the point and their boundary is the point's horizon. Identifying the visible facets can be facilitated by the above described direction decision when performing the detection if this new point is outside of the convex hull. Figure 3.6 shows the horizon of p_r on the $CH(P_{r-1})$. In Figure 3.7, consider plane h_f containing a facet f of $CH(P_{r-1})$, f is visible from a point if that point lies in the open half-space on the other side of h_f.

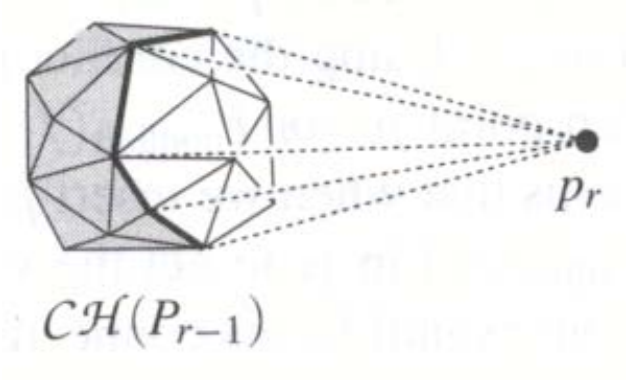


Figure 3.6 Visible Region of p_r on the $CH(P_{r-1})$

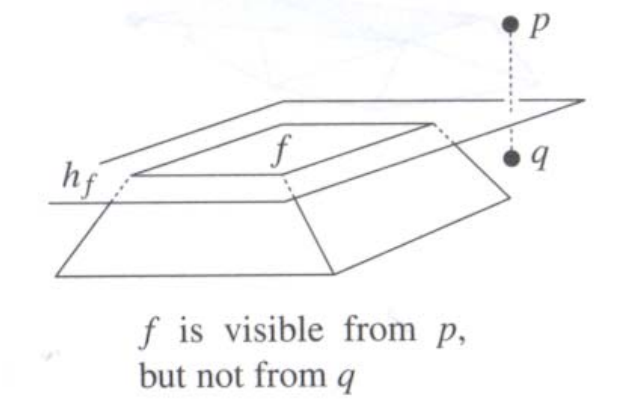


Figure 3.7 Visible Facet

Next, these visible facets are replaced by a cone of new facets. Each new facet is defined by the point and one horizon facet. Figure 3.8 shows the new convex hull $CH(P_r)$ by connecting each horizon edge with p_r to create a new triangle facet.

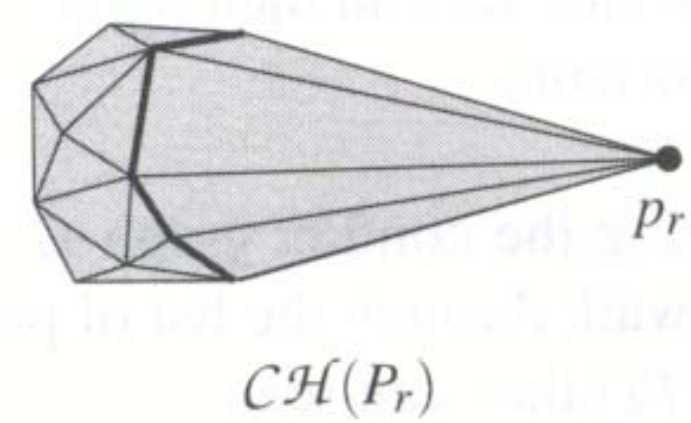


Figure 3.8 Convex Hull $CH(P_r)$

3.3 Reduction in Dimensionality

In this section we discuss the sensitivity analysis used to reduce the dimensionality of the data.

The first reason is due to computation effort mentioned in the last section. The other reason is because of the curse of dimensionality [49]. When a large number of features per object is used, it causes a severe overfitting problem, so increasing the number of features can deteriorate the classification performance.

However, sensitivity analysis is not the focus of this research. Details can be referred to in [50][51].

3.4 Comparison with Spherical Data Description

In this section, we make the comparison between our proposed convex hull approach and the spherical data description from the following data characteristics:

- Flexible description

The hypersphere is a very rigid model of the boundary of the data. In general, it cannot be expected that this model will fit the data well. By comparison, the convex hull can obtain a better fit between the actual data boundary and the hypersphere model.

- Sample size

Because faulty objects can be anywhere in the feature space, in any direction around the target data a boundary should be defined. When no strong model is assumed, the required number of training objects increases dramatically with the dimensionality of the data. We choose two different sizes of uniform distributed data. One contains 100 data points; the other one has 500 data points. For illustration, we only take two-dimensional data. From the boundary of the convex hull, it is obvious that the hull edge number doesn't increase with the sample size (Figure 3.9). It can be concluded that the complexity of the convex hull boundary does not vary with the increase of sample size.

In Equation (3.3) the free parameters a, R and ξ have to be optimized. Constraints (3.4) can be incorporated into the formula (3.3) by introducing Lagrange multipliers and constructing the Lagrangian:

$$L(R,a,\xi,\alpha,\gamma) = R^2 + C\sum_i \xi_i - \sum_i \alpha_i\{R^2 + \xi_i - (x_i \cdot x_i - 2a \cdot x_i + a \cdot a)\} - \sum_i \gamma_i \xi_i$$

$$\alpha_i \geq 0 \text{ and } \gamma_i \geq 0 \qquad (3.7)$$

L has to be minimized with respect to R, a and ξ, and maximized with respect to α and γ. Setting partial derivatives to 0 gives the constraints:

$$\sum_i \alpha_i = 1$$

$$a = \sum_i \alpha_i x_i$$

$$0 \le \alpha_i \le C$$

This results in the final error L:

$$L = \sum_i \alpha_i (x_i \cdot x_i) - \sum_{i,j} \alpha_i \alpha_j (x_i \cdot x_j) \tag{3.8}$$

The minimization of this error with the constraints is a well-known quadratic programming problem. The center of the sphere is expressed as a linear combination of objects with weights α_i. Only objects with positive weight are needed in the description of the data set. These objects are called the support objects of the description. By definition, R^2 is the (squared) distance from the center of the sphere a to one of the support objects of the description on the boundary:

$$R^2 = (x_k \cdot x_k) - 2\sum_i \alpha_i (x_i \cdot x_k) + \sum_{i,j} \alpha_i \alpha_j (x_i \cdot x_j) \tag{3.9}$$

for any $x_k \in SV^{bnd}$, i.e. the set of support objects of the boundary for which $0 < \alpha_k < C$.

From the formula above, it is noticed that these data objects with positive weights are needed for the boundary of data description. These weights are optimized by minimizing Equation (3.3). Therefore any data objects can be used as boundary points. However, in

Quickhull algorithm, it selects the furthest point of an outside set as the vertex of the convex hull [47]. Selecting a furthest point is important for bounding the maximum error.

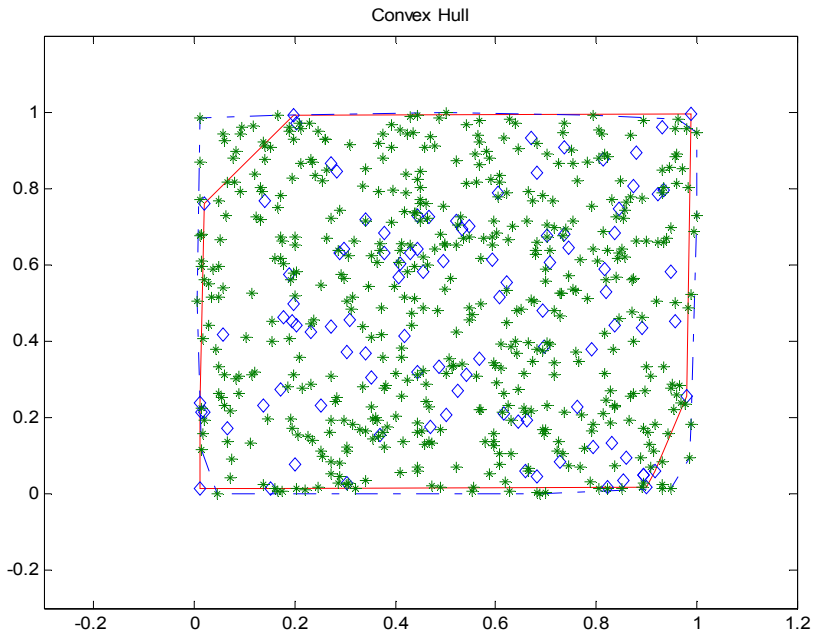


Figure 3.9 Convex Hull for Different Sizes of Samples

As for sphere data description in Figure 3.10, two circles represent the different boundaries of the data. The inner circle represents the tightest boundary; the outer circle is the loosest boundary. This technique allows the users to choose any boundary between these two circles by allowing different training errors. For small sized data, the variation range of boundaries is not significant. However, when the sample size becomes larger, the boundary range doesn't make sense any more (Figure 3.11). It indicates that the sphere data description is very sensitive to the sample size you choose. Furthermore, the training time is drastically extended with the increase of sample size compared to the training time of the convex hull.

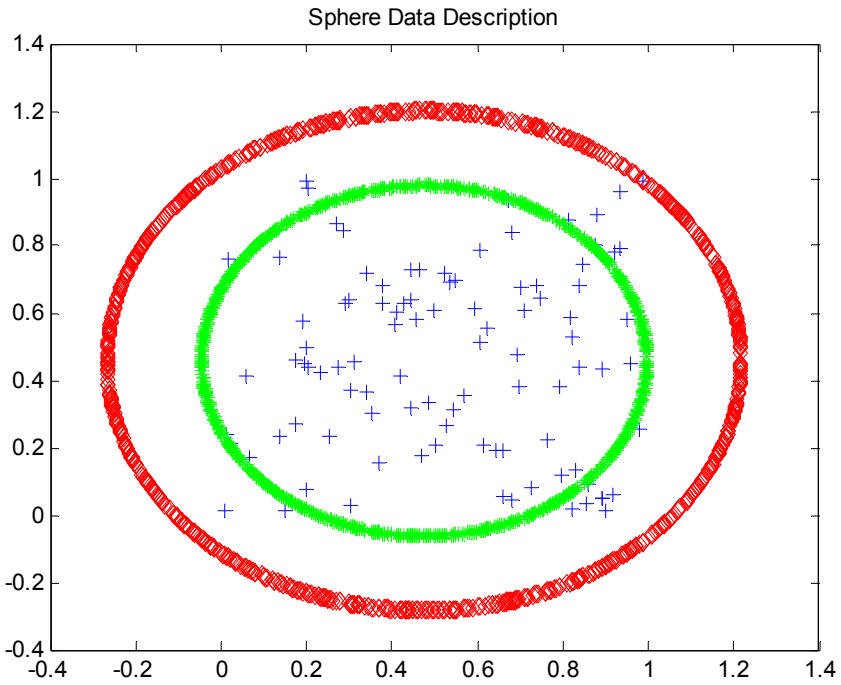


Figure 3.10 Sphere Data Description with Uniformed Sample Size of 100

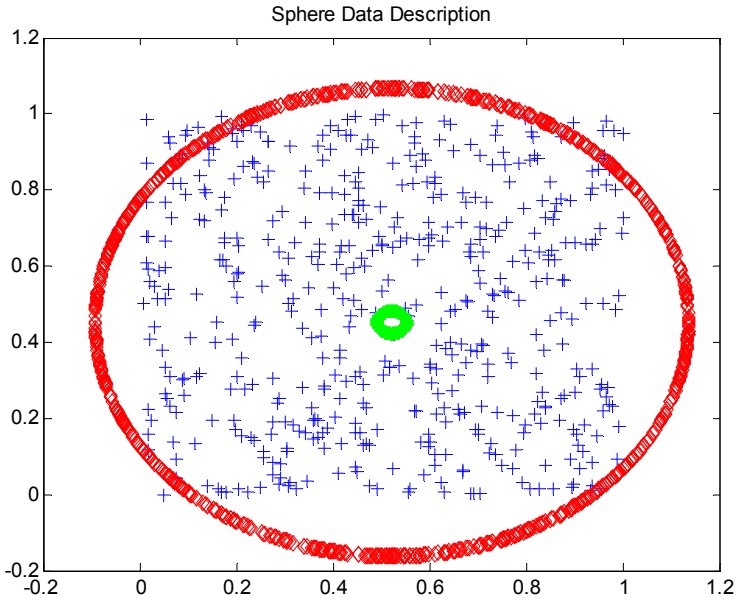


Figure 3.11 Sphere Data Description with Uniformed Sample Size of 500

- Scaling

Both techniques suffer from poorly scaled features in the experiment. Therefore prior to building the boundary, it is imperative to normalize data features.

- Non-convexity

The influence of the non-convexity is investigated by a banana-shaped dataset. This dataset has a non-convex distribution for the target class. Data is uniformly distributed in a half circle; the radial distribution is normally distributed. Figure 3.12 compares the convex hull boundary with a rigid circle from the hypersphere data description. It is noted that when the target class has non-convexity distribution both techniques can't give as tight a boundary as the one for convexity distribution. Therefore, in the beginning we already made it clear that we assume the normal class sensor data cluster in a convex set centered at the trimming values.

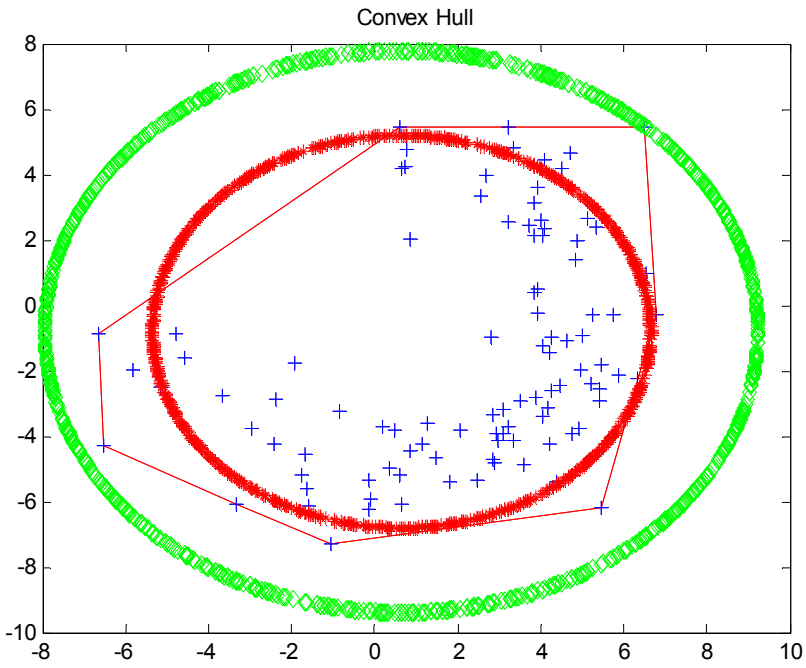


Figure 3.12 Convex Hull and Sphere Data Distribution for Non-Convex Distribution

Chapter 4

Fault Trending Analysis Part II – Support Vector Machine

4.1 Two-Class Classification

In Chapter 3, we discussed the one-class classification for fault detection by determining the boundary of the normal target class. When the normal data has non-convex distribution, the boundary decided by only one class may not be tight enough to label an unknown element in the correct class. We make a simple example to illustrate this point. In Figure 4.1, a two-class classifier and one-class classifier are applied to an artificial dataset representing two features. The normal data set has a banana-shaped distribution which is non-convex. The thin solid line describes the normal dataset boundary with a one-class classifier. By contrast, when faulty class data is used at the same time, we could have the thick solid line as the two-class classifier that distinguishes between the normal and faulty classes. One can notice that the boundary of the two-class classifier is more tight and accurate than the one-class classifier. For an unknown class data shown in the figure, one-class classifier assigns it to the faulty class; the two-class classifier labels it as the normal class. In this chapter, we consider building the decision boundary between two classes with a two-class classifier assuming that faulty class data is available.

A two-class classifier is a function $f(x): R^N \rightarrow \{-1,+1\}$ that outputs a class label -1 or $+1$ for each input object $x \in R^N$. If the classifier cannot be constructed from any known rules, one tries to infer a rule from a limited set of training examples. There are well known inherent risks in this approach [52]; the set of training examples might be very uncharacteristic; the inherent variance in the objects and noise in the measurements

might be too big to extract classification rules with high confidence. The smaller the number of training examples, the more pronounced this problem becomes [52].

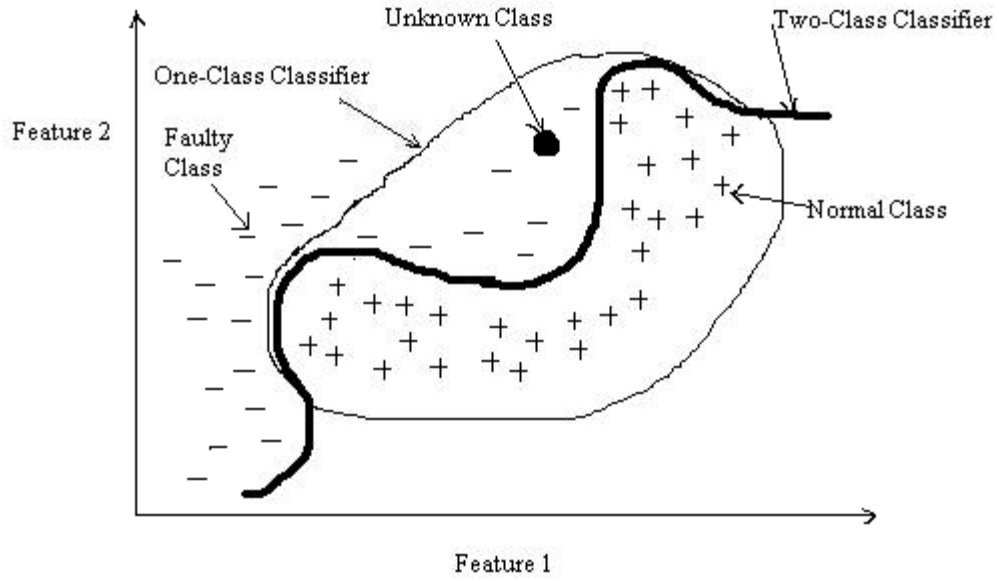


Figure 4.1 A Two-Class Classifier and One-Class Classifier

Even when good characteristic samples are available, the number of functions that approximates or precisely fits the data may be very big. For this reason, most often the type of function f is chosen beforehand and a few parameters w of the function have to be determined. For a set of training data $X := \{x_1, x_2, ..., x_m\} \subseteq R^d$, where $m = N$ corresponding labels $Y := \{y_1, y_2, ..., y_m\} \to \{-1, +1\}$ is attached. The function f should be constructed so that for a given feature vector x an estimate of the label is obtained, $y = f(x, w)$, $f(x; w) : R^d \to \{-1, +1\}$. The optimal parameters w for the function f on a given training data set X is defined as [52]:

$$w = \arg\min \varepsilon_{true}(f, w, X) \tag{4.1}$$

$\varepsilon_{true}(f, w, X)$ is the expectation of the test error for the trained function $f(x, w)$.

In the computation of ε_{true}, it requires the integration over the complete probability density $p(x, y)$ of all possible objects x and labels y. An induction principle has to be adopted to approximate the true error [52].

$$\varepsilon_{true}(w) = \int |y - f(x, w)| p(x, y) dx dy \tag{4.2}$$

In almost all classification problems this $p(x, y)$ will be unknown. It is hoped that the training set is a representative sample from this true distribution, but in most cases this will not be the case. In practice ε_{true} is often approximated by the empirical error of the training set $\varepsilon_{emp}(f, w, X)$. Most often the objects in the training data are assumed to be independently distributed, so the total empirical error of function f on a training set is defined as:

$$\varepsilon_{emp}(w) = \frac{1}{N} \sum_{i=1}^{N} |y_i - f(x_i, w)| \tag{4.3}$$

$\varepsilon_{emp}(w)$ is a fixed number for a particular choice of w and for a particular training set $\{x_i, y_i\}$. The quantity $|y_i - f(x_i, w)|$ can only take the values of 0 and 1.

The empirical error gives an approximation of the true error, which becomes accurate when the training data is distributed like the true data distribution and the sample size is very large. In general, the larger the sample size, the better the characteristics of the data can be determined. Unfortunately, when it is used to optimize $f(x, w)$, a low empirical error does not guarantee a low true error. The phenomenon that a function $f(x, w)$ minimizes the empirical error very well on a training set but still shows a large true error on an independent test set is called overfitting [42]. Therefore, good

classification of the training objects is not the main goal, but good classification of new and unseen object is. This is called good generalization. In section 4.3, we will give the analysis of the estimation of the proposed classifier generalization. The overfitting of a classifier function $f(x)$ is illustrated with a conceptual example shown in Figure 4.2. Here a very flexible function is trained on the data. Because the function is far too flexible for this data, it finds structure in the data that is not really there. The overtrained classifier suggests that the left class actually consists of two separate clusters. However, these two clusters should belong to one class. Therefore, testing this classifier with independent test data will reveal that the generalization performance is not very high. Moreover, this overfitting problem becomes more severe when large numbers of features are used [42].

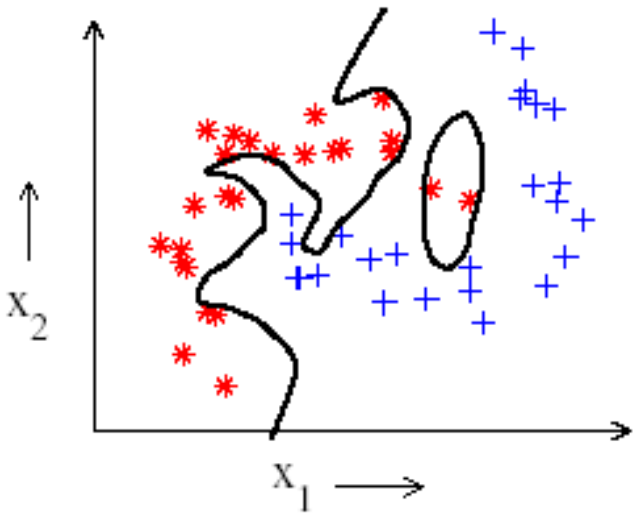


Figure 4.2 An Overtrained Classifier

Hence, the main goal in this chapter is to find a classifier that shows good generalization in classifying unseen objects.

4.2 Support Vector Machine

The support vector machine (SVM) is a supervised learning algorithm first introduced by Vapnik [52] for pattern recognition. Given a set of labeled training vectors (positive and negative input examples), each object x_i attached with a label y_i ($y_i \in \{-1,+1\}$)), SVM [54][55][56] build a linear decision boundary to discriminate between the two classes. SVM has recently become an area of intense research owing to developments in the techniques and theories coupled with extensions to regression and density estimation [54].

In binary classification we are given a set of training vectors with its labels $\Theta = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$, $x_i \in R^d$, $y_i \in \{-1,1\}$. The task is to train a SVM classifier to learn a relationship between the data and its respective labels. SVM is a non-linear classifier defined with the linear technique thanks to a non-linear mapping function. The geometrical interpretation of SVM is that the algorithm searches for the two optimal parallel separating surfaces, i.e. the hyperplanes, which have maximum distance from the two classes. SVM is outlined first for the linearly separable case in the following. The kernel trick is then introduced in order to construct non-linear decision surfaces for non-linearly separable case.

1. The Separable Case

SVM is capable of learning linear hyperplanes to discriminate between the two classes (Figure 4.3). The distance from the hyperplane to the closest positive (negative) example is known as the margin of the hyperplane. Maximizing the margin of the hyperplane is then equivalent to maximizing the distance between the class boundaries. The linear

hyperplane can be defined as: $f(x) = \langle w, x \rangle + b$. The weight vector w is a normal vector to the hyperplane and b is the bias.

Find $f(x) = (w^T x + b)$ with maximal margin, such that [56]

$$f(x_i) = \langle w, x_i \rangle + b \geq 1, \text{ for } y_i = 1$$

$$f(x_i) = \langle w, x_i \rangle + b \leq -1, \text{ for } y_i = -1 \tag{4.4}$$

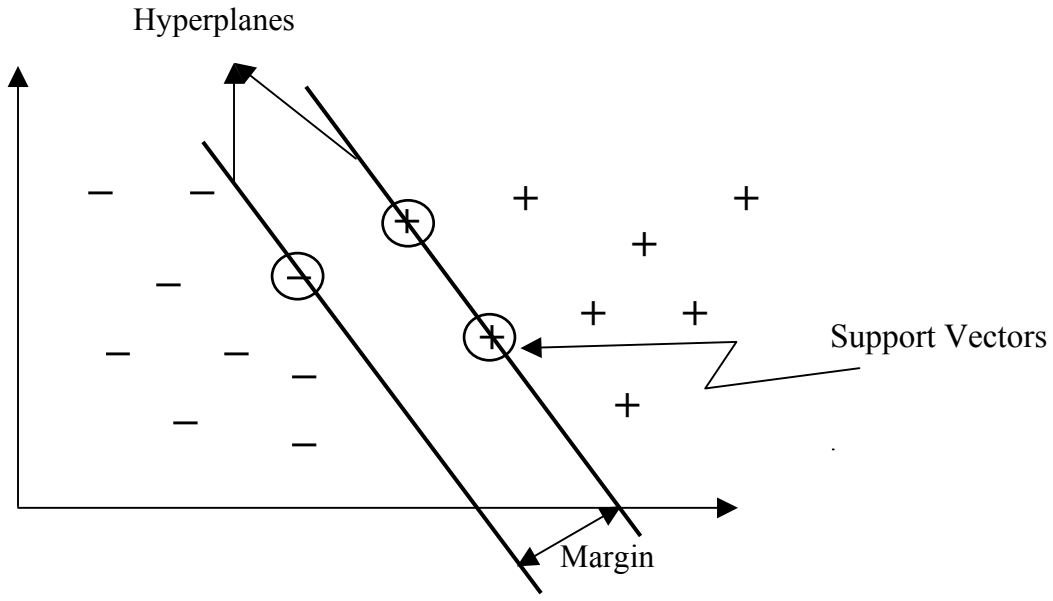


Figure 4.3 Support Vector Machine

Equation (4.4) can be combined into one set of inequalities:

$$y_i (x_i \cdot w + b) - 1 \geq 0, \ \forall i \tag{4.5}$$

The task of learning coefficients w and b of support vector machine $f(x) = (w^T x + b)$ is reduced to solving the following constrained optimization problem [56]:

Find **x** and b that minimize: $\frac{1}{2}\|w\|^2$

Subject to $y_i(x_i \cdot w + b) - 1 \geq 0,\ \forall i$ (4.6)

We introduce positive Lagrange multipliers α_i, $i = 1,...,l$, one for each of the above inequality constrains. Then the above optimization problem can be solved by using the Lagrangian function defined as a primary Lagrangian [56]:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} - \sum_{i=1}^{N}\alpha_i[y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) - 1]$$

Such that $\alpha_i \geq 0, \forall i$ (4.7)

The necessary conditions for the saddle point of $L(\mathbf{w}, b, \alpha)$ are

$$\frac{\partial L}{\partial w_j} = 0, \quad \forall j$$
$$\frac{\partial L}{\partial \alpha_i} = 0, \quad \forall i$$

Solving for the necessary conditions results in

$$w = \sum_{i=1}^{N}\alpha_i y_i x_i$$
$$\sum_{i=1}^{N} a_i y_i = 0$$

By replacing $w = \sum_{i=1}^{N}\alpha_i y_i x_i$ into the Lagrangian function and by using $\sum_{i=1}^{N} a_i y_i = 0$ as a new constrain the dual optimization problem can be constructed as[56]:

Find α that maximizes $\sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j \mathbf{x}_i^{T}\mathbf{x}_j$

Subject to $$\sum_{i=1}^{N} a_i y_i = 0, \quad \alpha_i \geq 0, \quad \forall i \qquad (4.8)$$

This is a convex quadratic programming problem, so there is a global minimum. Notice that there is a Lagrange multiplier α_i for every training point. In the solution, those points for which $\alpha_i > 0$ are called "support vectors", and lie on one of the hyperplanes. Given the values α_1, α_2, … α_N obtained by the solution of the dual problem, the final SVM predictor can be expressed as [54]:

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}_i + b = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i{}^T \mathbf{x} + b$$

Where $$b = \frac{1}{\left|I_{\text{support}}\right|} \sum_{i \in I_{\text{support}}} \left(y_i - \sum_j \alpha_j y_j \mathbf{x}_j{}^T \mathbf{x}_i \right)$$

And I_{support} is the set of support vectors.

2. The Non-Separable Case

The above linear technique for separable data can be applied to non-separable data by mapping the data into a feature space using a non-linear mapping function where they are linearly separable. After introducing positive slack variable ζ_i in the constraints, Equation (4.4) becomes [56]:

$$f(x_i) = \langle w, x_i \rangle + b \geq 1 - \xi_i, \text{ for } y_i = 1$$

$$f(x_i) = \langle w, x_i \rangle + b \leq -1 + \xi_i, \text{ for } y_i = -1 \qquad (4.9)$$

The optimization problem for construction of SVM is defined as:

Find **x** and b that minimize: $\frac{1}{2}\|\mathbf{w}\|^2 + C\sum_i \xi_i{}^2$

$$\text{Subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad \forall i \tag{4.10}$$

Likewise, this optimization problem can be solved by using the Lagrangian function defined as a primary Lagrangian [56]:

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w} - \sum_{i=1}^{N}\alpha_i[y_i(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + b) - 1 + \xi_i] + C\sum_i \xi_i - \sum_i \mu_i\xi_i$$

$$\text{Such that } \alpha_i \geq 0, \forall i \tag{4.11}$$

Where $\alpha_1, \alpha_2, ..., \alpha_N$ are Lagrange multipliers and $\alpha = [\alpha_1, \alpha_2, ..., \alpha_N]^T$.

For the primal problem above, all KKT conditions (which are necessary for the optimal solution) may be stated [56]:

$$\begin{aligned}
&\frac{\partial L_P}{\partial w} = w - \sum_i \alpha_i y_i x_{iv} = 0 \\
&\frac{\partial L_P}{\partial b} = -\sum_i y_i \alpha_i = 0 \\
&\frac{\partial L_P}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \\
&y_i(\Phi(x_i) \cdot w + b) - 1 + \xi_i > 0 \\
&\xi_i > 0 \\
&\alpha_i > 0 \\
&\mu_i \geq 0 \\
&\alpha_i\{y_i(\Phi(x_i) \cdot w + b) - 1 + \xi\} = 0 \\
&\mu_i\xi_i = 0
\end{aligned} \tag{4.12}$$

By replacing $w = \sum_i \alpha_i y_i x_{iv}$ into the Lagrangian function and by using $\sum_i y_i\alpha_i = 0$ as a new constraint this optimization problem can be converted to its dual problem [56]

$$\text{Find } \alpha \text{ that maximizes} \qquad \sum_i \alpha_i - \frac{1}{2}\sum_i\sum_j \alpha_i\alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$$

$$\text{Subject to} \quad \sum_{i-1}^{N} \alpha_i y_i = 0 \qquad (4.13)$$
$$0 \le \alpha_i \le C, \quad \forall i$$

The resulting SVM is of the form

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x}_i + b = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i{}^T \mathbf{x} + b$$

$$b = \frac{1}{\left| I_{\text{support}} \right|} \sum_{i \in I_{\text{support}}} \left(y_i - \sum_j \alpha_j y_j \mathbf{x}_j{}^T \mathbf{x}_i \right) \qquad (4.14)$$

I_{support} is the set of support vectors.

It is noticed that the way in which the data appears in the training problem (4.14) is in the form of dot products $(x_i \cdot x_j)$. First we map the data to some other Euclidean space H, using a non-linear mapping function $\Phi : R^d \to H$, $x \to \Phi(x)$. Then the training algorithm would only depend on the data through dot products $\Phi(x_i) \cdot \Phi(x_j)$. We can define a positive definite kernel via the kernel trick which regards the kernel as generalized dot products:

$$K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j) \qquad (4.15)$$

In the test phase, an SVM is then used by computing the sign of $f(x)$ [57].

$$f(\mathbf{x}) = \sum_{i=1}^{N_s} \alpha_i y_i K(s_i, x) + b \qquad (4.16)$$

where s_i is the support vector so that we can use the $K(s_i, x) = \Phi(s_i) \cdot \Phi(x)$. Most importantly, we are still doing the linear separation, but in a different space by the help of a non-linear mapping function $\Phi(x)$. Most common used kernel functions in SVM are

polynomial kernel $K(x,y)=(x,y)^d$ and Gaussian radial basis function (RBF) kernel $K(x,y)=\exp(-\frac{\|x-y\|^2}{2\sigma^2})$. In the next section, we derive subsequence-based kernel by starting to define an explicit mapping $\Phi: R^d \rightarrow H$.

4.3 Subsequence-Based Kernel Function

Our data can be classified as time series data, i.e. a sequence of real numbers, each number representing a value at a time point. A big problem when classifying in time series data is the high dimensionality [58]. It is usually not enough to look at each point in time sequentially; rather one has to deal with sliding windows of a possibly multi-dimensional time series. This quickly produces very high dimensional vectors, introducing the so-called curse of dimensionality and causing problems with distance metrics [59]. Moreover, consecutive values of a time-series are usually not independent but highly correlated, thus there is a lot of redundancy. Therefore using all time points is not really necessary. Solutions to discovering knowledge in time-series database in early work can be classified into two categories.

- The time-domain-based approach

This approach handles time-series data in the time-domain using shifting, scaling, smoothing and time warping methods [60]. This approach is complex and results in low performance because of focusing on every single data point.

- Transformation-based representation

This approach transforms the initial sequence from time to another domain, and then uses a point in the new domain to represent each original series. A well-established feature extraction technique is to use DWT or DFT for time series. It uses only the *k* largest

coefficients of each time series because they preserve the optimal amount of energy per time series [61]. It is less controllable by a user, therefore it is not well suited for finding similar patterns with time-series data sequences [62].

To solve the drawbacks stated in the above two approaches, what we propose is another time-domain approach. Our proposed method aims to develop a sequence classification method in the context of SVM. Subsequence-based kernel function is developed for this purpose. The starting point is to focus on the concept of informative subsequences (i.e. sequences of meaningful data points without redundant information) rather than all data points. Non-uniform quantization is developed for highlighting the informative subsequences. Then the subsequence kernel function is designed to measure the similarity of any two strings. If two strings contain many of same subsequences, their kernel will be large.

So far, the subsequence-based kernel for discrete sequences was used for text classification [63] and in the analysis of DNA sequences. The main idea of their subsequence kernel is to compare strings not by words, but by the substrings they contain. These substrings do not need to be contiguous, but they receive different weighting according to degree of contiguity [63].

4.3.1 Non-Uniform Quantization

Our data is a continuous time series. To develop a subsequence-based kernel, we have to do the quantization first. An analog value is normalized with respect to a defined range and discretized into bins. Each data is then assigned a symbol corresponding to the bin in which it falls. The size of the bins is determined by the number of bits used in the discretization. If each data is encoded by m bits (which may be chosen according to the

desired precision), then there would be 2^m different bins between the maximum and minimum ranges of data. Figure 4.4 shows 10 normal real-valued simulation runs of roll angle, each simulation having 3000 data points. Its corresponding histogram with $m = 4$ (16 bins) is shown in Figure 4.5.

If the input s is uniformly distributed over a finite interval $[A, B]$, then the output of a uniform quantizer $Q(s)$ with fixed level L is given by:

If $s \in (d_i, d_{i+1}]$, then

$$Q(s) = r_i = d_{i-1} + \frac{q}{2},\ 1 \le i \le L$$

$$\text{With} \quad q = \frac{B - A}{L},\ d_i = A + iq \ \text{ for } 0 \le i \le L \tag{4.17}$$

s is the continuous input. d_i and r_i are the decision and reconstruction levels, respectively.

However, in Figure 4.5 of the data histogram, it is noticed that the data doesn't have uniform distribution at all! Actually data is sparsely distributed in range [-0.05 – 0.02] and [0.015 0.03]. It indicates that the data has very low likelihood of staying in those ranges. In other words, the characteristic of the normal data is hidden in the dense region. In the case of non-uniform distribution, a non-uniform quantization scheme should be derived. Figure 4.6 illustrates one definition of non-uniform quantization.

We want subtle discretization in the dense region instead of sparse in order to avoiding losing important information. This procedure is called non-uniform quantization. Specifically, we can assign the same symbol to sparse distributed regions

that are contiguous. As for the data in the dense distributed region, we can subdivide them into smaller regions, labeling data within each smaller region with the same symbol.

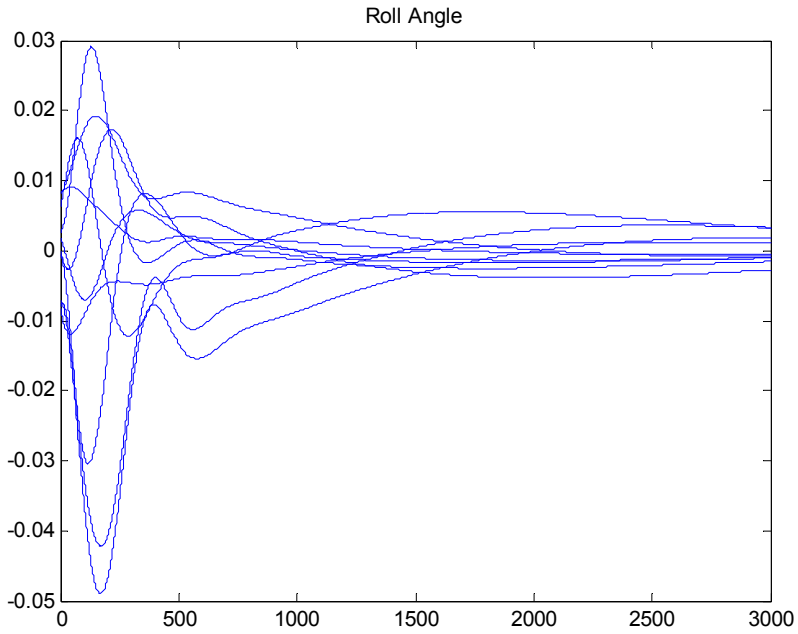


Figure 4.4 Real-Valued Data

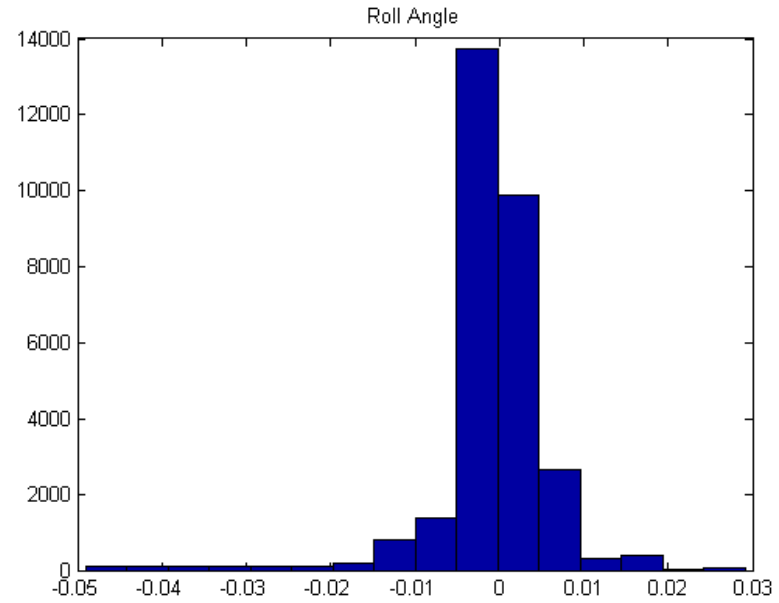


Figure 4.5 Histogram of Real-Valued Data

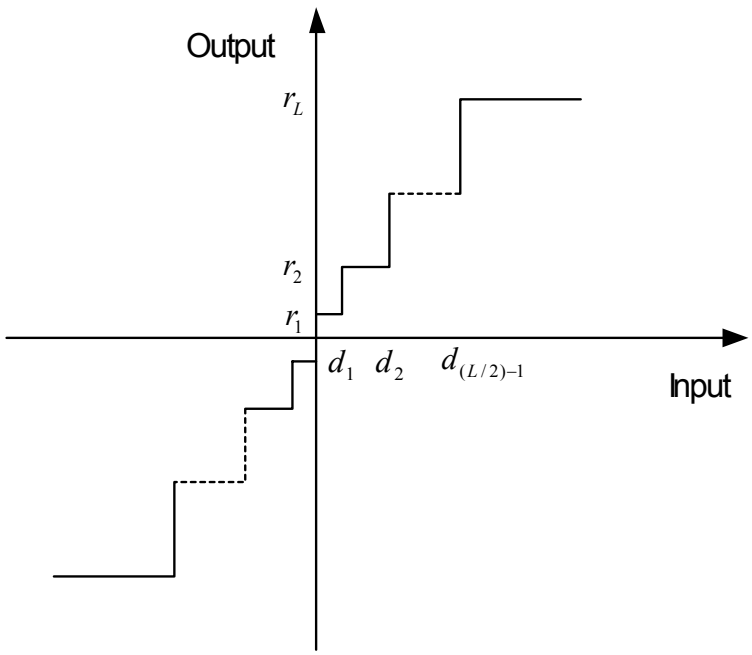


Figure 4.6 Non-uniform Quantization

Here the Lloyd quantization algorithm is applied [64]. The basic idea of Lloyd quantization is to select the appropriate decision level d_i and reconstruction level $r_i\,(i=1,...,L)$ in order to minimize the mean square quantization error D with respect to d_i and r_i.

$$D = E\{(s-r_i)^2\} = \sum_{i=1}^{L} \int_{d_{i-1}}^{d_i} (s-r_i)^2\, p(s)ds \tag{4.18}$$

$p(s)$ is a given input probability density function.

The solution results in decision levels and reconstruction levels as:

$$d_i = \frac{r_{i-1}+r_i}{2},\ 1 \le i \le L-1$$

$$r_i = \frac{\int_{d_{i-1}}^{d_i} sp(s)ds}{\int_{d_{i-1}}^{d_i} p(s)ds},\ 1 \le i \le L \tag{4.19}$$

In general, Equation (4.19) does not yield closed-form solutions. Therefore they need to be solved by numerical techniques. When a numerical solution is necessary, the following iterative algorithm can be used. First, an arbitrary initial set of values for d_i is chosen, and the optimum r_i for that set are found by using Equation (4.19). For the calculated r_i, the optimum d_i are then determined using Equation (4.19). This process is iterated until the difference between the two successive approximations is below a threshold. In the Matlab toolbox, there is a built-in function Lloyds provides r_i, d_i, D and the relative change in the distortion D between the last two iterations.

We can determine the bits number by checking when the relative change in distortion between iterations becomes constantly small. Figure 4.7 shows the distortion D with respect to the number of bits L for the normal data. Figure 4.8 ~ Figure 4.11 shows the quantization error D against the number of bits L for the engine fault, the aileron fault, the elevator fault, the rudder fault and the stabilizer fault. It is observed that when $m \ge 4$ relative change in distortion D is unnoticeable for all faulty data and normal data. Therefore, $m = 4$ is chosen as the bits number for discretization, that is, there are 16 bins between the maximum and minimum ranges of data.

Figure 4.12 compares the continuous normal data with piecewise approximation via 16-level quantization. Figure 4.13 does the same thing for engine fault data. Figure 4.14 and Figure 4.15 provide the non-uniform histogram for normal and engine faulty data. It is noticed that the smaller quantization step is used in the densely distributed region and the bigger step size is for in the sparsely distributed region.

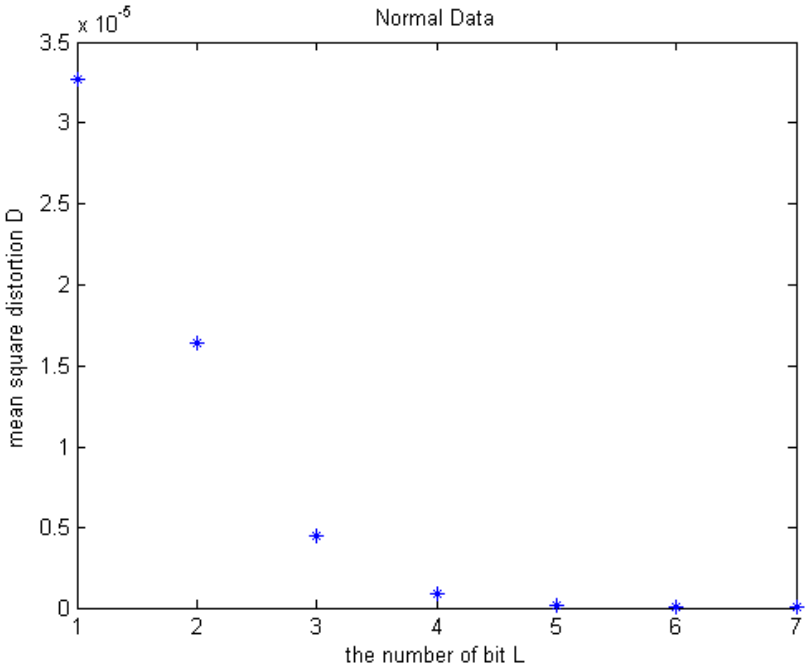


Figure 4.7 Mean Square Distortion vs. Bits Number for Normal Data

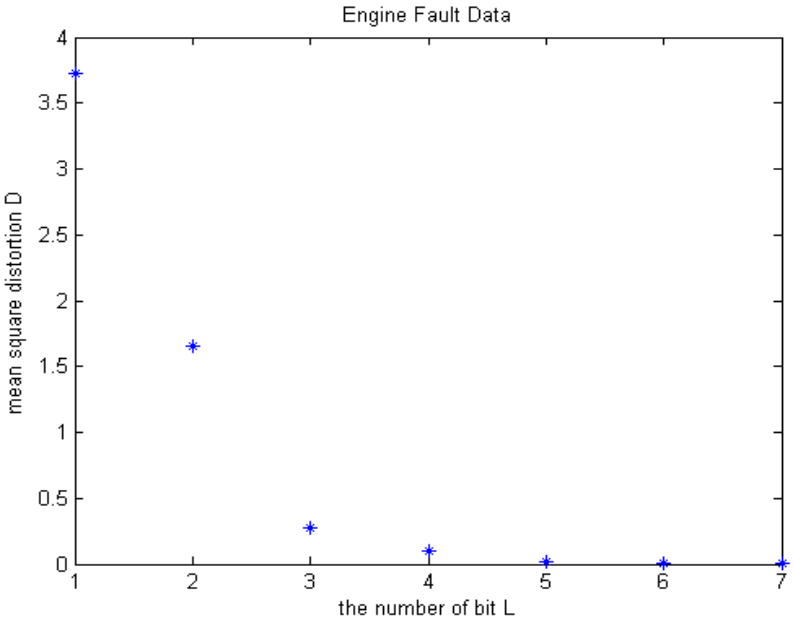


Figure 4.8 Mean Square Distortion D vs. the Number of Bits for the Engine Fault

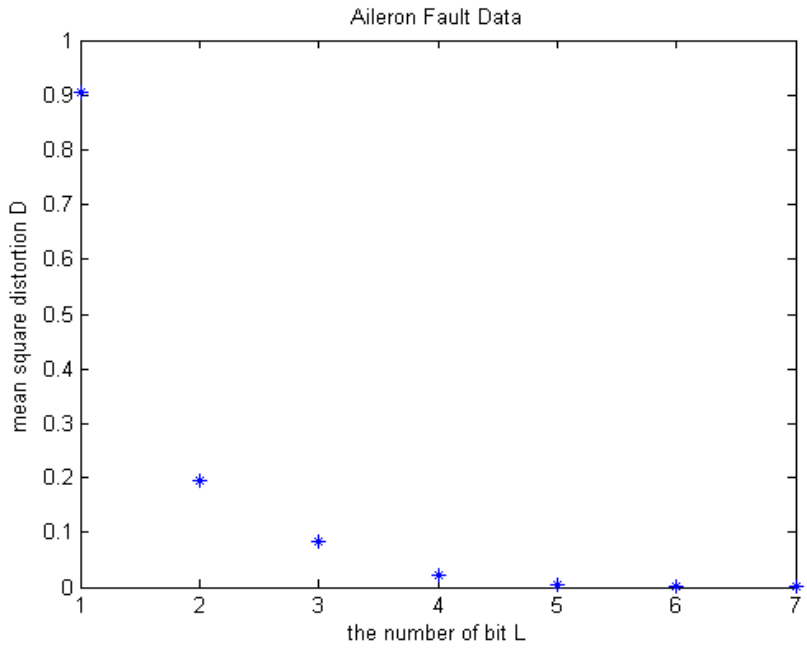


Figure 4.9 Mean Square Distortion D vs. the Number of Bits for the Aileron Fault

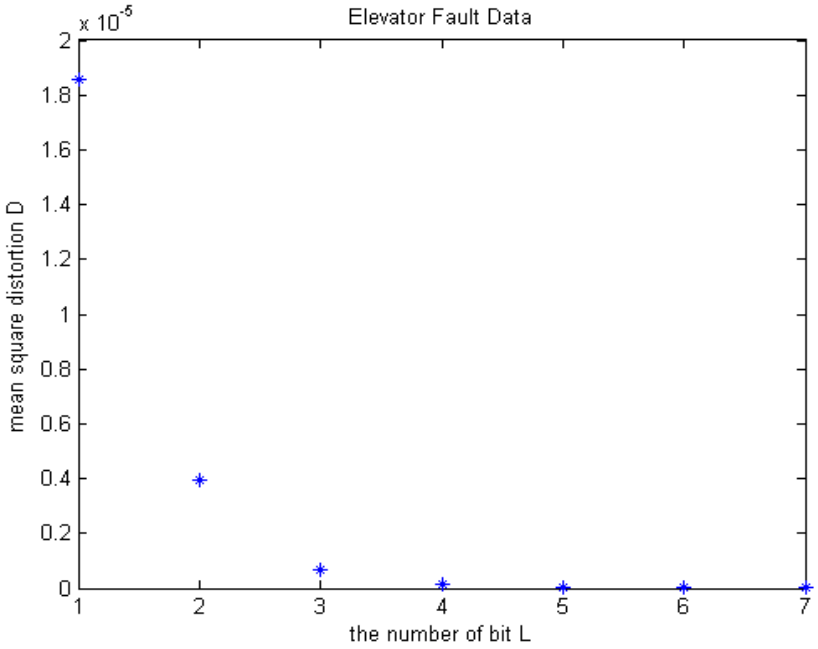


Figure 4.10 Mean Square Distortion D vs. the Number of Bits for the Elevator Fault

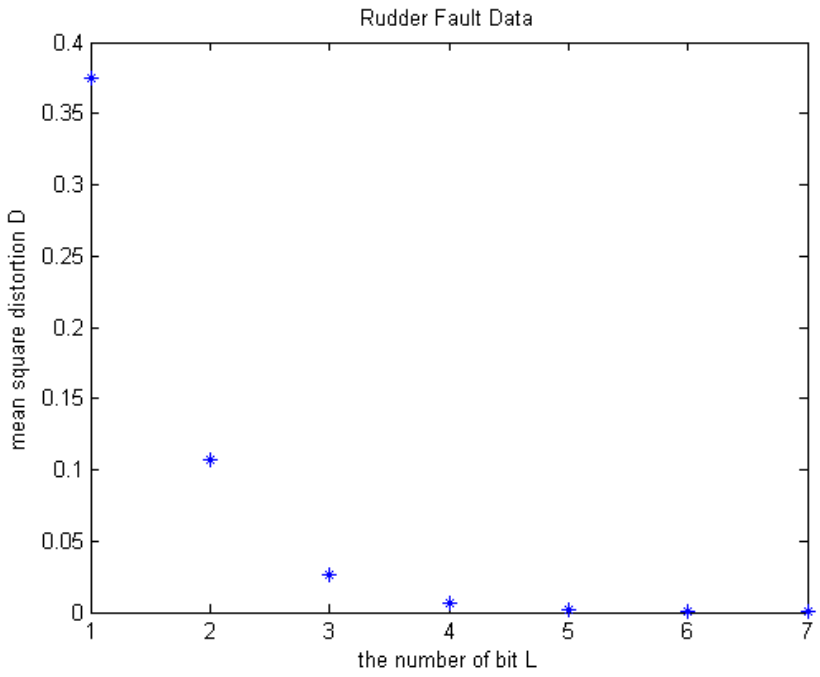


Figure 4.11 Mean Square Distortion D vs. the Number of Bits for the Rudder Fault

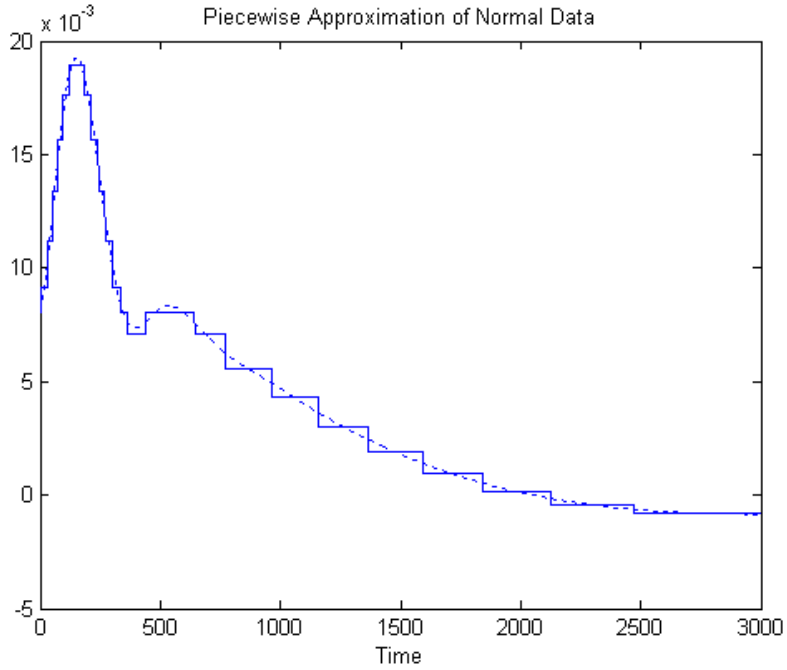


Figure 4.12 Piecewise Approximation of Normal Data

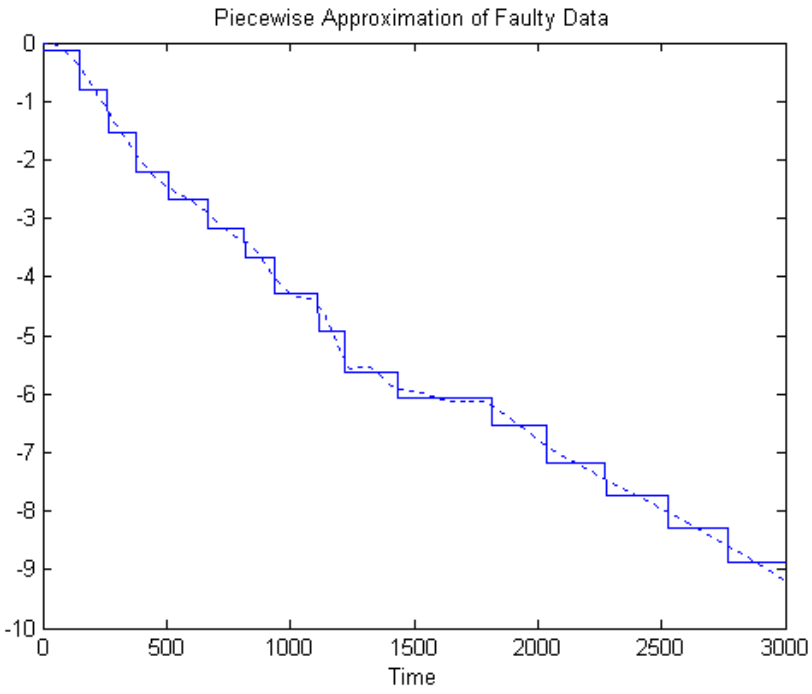


Figure 4.13 Piecewise Approximation of Engine Fault Data

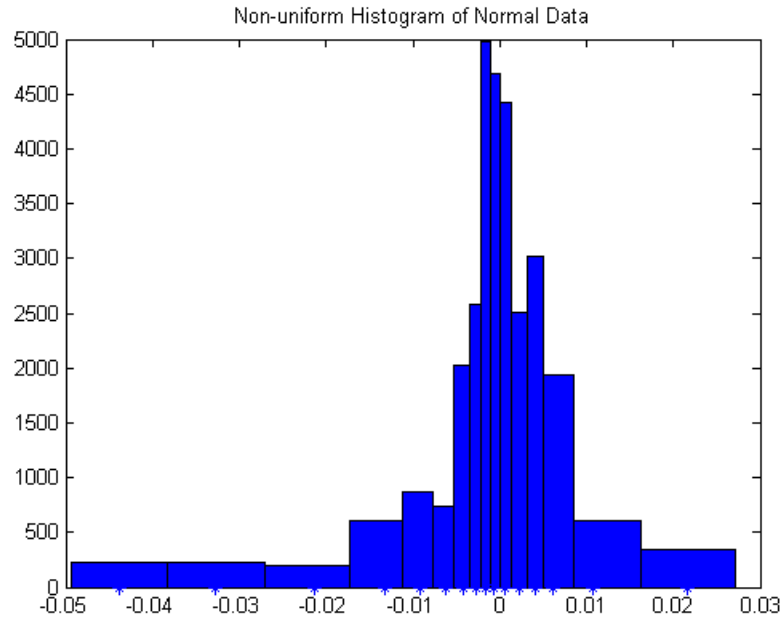


Figure 4.14 Non-Uniform Histogram of Normal Data

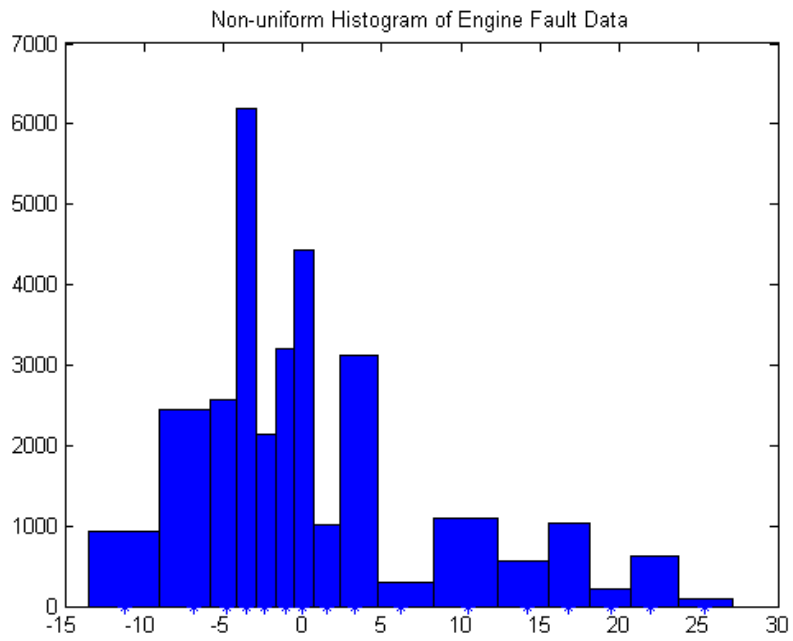


Figure 4.15 Non-Uniform Histogram of Engine Fault Data

By minimizing the mean square quantization error D, we select a set of decision and reconstruction levels d_i and r_i for best approximating continuous data. To make a string, the final step is to assign different symbols to reconstruction levels r_i. For the example in Figure 4.14, 'a' is assigned to r_1 corresponding to d_1 which roughly spans over [-0.05, -0.04]. 'b' is assigned to r_2, and so on. At this stage, the continuous numeric data has been transformed into discrete character strings. In the following, normal and faulty data patterns can be extracted from these character strings.

4.3.2 Subsequence Library

After converting the time series data to a string, we need to build a subsequence library for storing normal and faulty patterns (subsequences). In this section, we construct a

subsequence library recording all possible k-length contiguous subsequences in normal and faulty training strings.

In the non-uniform quantization of the normal data, associated symbols from the defined normal symbol set are assigned to each normal data, so that all normal data becomes a long string. A sliding window of k-length is moving across this long string. All different k-length subsequences present in the normal string are stored into the library. The same procedure applies to faulty strings. In the end, we combine all those subsequences into the k-length subsequence library. This k-length subsequence library is a vector with each element as a k-length subsequence appearing in normal or faulty training strings.

$$L = \{l_i\}, \; i = 1,...,n \tag{4.20}$$

l_i is a k-length subsequence.

Determination of a subsequence library requires the definition of the length for a subsequence and a step size for two sliding windows. Determination of these parameters demands extensive experimental work. The effect of each is described below.

First of all, the length of a subsequence depends on the number of quantization levels. Since we do not want to lose any valuable information in performing non-uniform quantization, subtle quantization is preferred in densely distributed regions. However, subtle quantization is sensitive to the small data variation.

For example, for a normal training string ‘aaabbbbbccccabababbbbccccccaaabcbbbcbcc’,

If $k = 3$, subsequence library L = [‘aaa’, ‘aab’, ‘abb’, ‘bbb’, ‘bbc’, ‘bcc’, ‘ccc’, ’cca’, ‘cab’, ‘aba’, ‘bab’, ‘caa’, ‘abc’, ‘bcb’, ‘cbb’, ‘cbc’].

If $k = 9$, subsequence library L = [‘aaabbbbbc’, ‘aabbbbbcc’, ‘abbbbbccc’, ‘bbbbbcccc’, ‘bbbbccccа’, ‘bbbccccab’, ‘bbccccaba’, ‘bccccabab’, ‘ccccababb’, ‘ccababbbb’, ‘cababbbbc’, ‘ababbbbcc’, ‘babbbbccc’, ‘abbbbcccc’, ‘bbbbccccc’, ‘bbbccccca’, ‘bbcccccaa’, ‘bcccccaaa’,’ cccccaaab’, ‘ccccaaabc’, ‘cccaaabcb’, ‘ccaaabcbb’, ‘caaabcbbb’, ‘aaabcbbbc’, ‘aabcbbbcb’, ‘abcbbbcbc’, ‘bcbbbcbcc’].

The first scenario: we have two normal test strings which are

x_1 = ‘ababbbbbcccc’
x_2 = ‘ababbbcbcccc’

Their corresponding feature vectors according to the subsequence library $k = 3$ are

$\Phi(x_i)$ = [0 0 1 3 1 1 2 0 0 1 1 0 0 0 0 0]
$\Phi(x_j)$ = [0 0 1 1 1 1 2 0 0 1 1 0 0 1 0 1]

(The process calculating the feature vector is discussed in the next section).

As defined in Equation (4.15), the kernel function $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j)$. $\Phi(x)$ is the feature mapping function. Therefore, the kernel function for the above example is $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j) = 2$

Similarly, feature vectors according to the subsequence library $k = 9$ are

$\Phi(x_i)$ = [0 0 2 4 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
$\Phi(x_j)$ = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0].

But their kernel function $K(x_i, x_j) = \Phi(x_i)^T \Phi(x_j) = 0$.

Although two test strings x_1 and x_2 both belong to the normal class, the kernel value of $k = 9$ is significantly smaller than the one for $k = 3$. As mentioned earlier, the kernel is defined to measure the similarity between two strings. The bigger the kernel

value of the test string with normal training string is, the more similar they are and the more likely the test string is labeled as normal. The same principle applies to faulty training string. Therefore, in terms of the variance problem, the kernel calculated with the longer subsequence can give rise to the misleading classification. In comparison, a shorter subsequence can easily handle this data variation problem.

The second scenario is that two strings are from different classes. Since they have much more different subsequences than strings from the same class, it doesn't result in any significant difference by choosing either a shorter or longer length subsequence. Therefore, as the result of data variation we choose the shorter length subsequence to build the normal and faulty library.

In the following, we use B747 simulation data to compare different subsequence lengths and step size in terms of three factors: missed alarm rate, false alarm rate and time delay.

- Different subsequence length and step size does not have any impact on time delay.
- For a specific subsequence length, different step size does not make any difference for most faults in terms of missed alarm rate, false alarm rate.
- False alarm rate rises a little with larger subsequence length; by contrast, the missed alarm rate decreases a bit or keeps constant.

Table 4.1 The Comparison of Different Subsequence Lengths

Faults	Subsequence length	Step Size	False Alarm Rate	Missed Alarm Rate	Time Delay (sec)
Engine Failure	3	1	8.33%	1.67%	2.04
		3			
	5	1	8.33%	0.33%	2.02
		5			
	10	1	8.67%	0.33%	2.02
		10			
	20	1	9%	0%	2.02
		20			
Aileron Deflection	3	1	9%	1.67%	2.04
		3			
	5	1	9.33%	1.67%	2.04
		5			
	10	1	10%	1.67%	2.04
		10			
	20	1	10.33%	1.67%	2.04
		20			
Rudder Deflection	3	1	12.17%	0.17%	2.02
		3			
	5	1	12.33%	0.17%	2.02
		5			
	10	1	12.67%	0.33%	2.02
		10			
	20	1	14.67%	0	2.02
		20			
Stabilizer Deflection	3	1	27.33%	7.67%	2.02
		3			
	5	1	27.67%	7.67%	2.02
		5			
	10	1	27.67%	7.67%	2.02
		10			
	20	1	25.67%	12%	2.02
		20			

(Table continues.)

Elevator Deflection	3	1 3	25.33%	10.33%	2.02
	5	1 5	25.67%	9.58%	2.02
	10	1 10	26.67%	10%	2.02
	20	1 20	22.67%	13%	2.02

The second effect is the amount of feature patterns. A longer subsequence results in a higher number of possible patterns in the library. In the example above, $k = 9$ subsequence has 28 normal feature patterns in the library; $k = 3$ subsequence has 16 normal feature patterns in the library. After adding faulty feature vectors in, the library size is even higher. It results in the computation complexity problem that causes the longer time delay in the real time fault detection.

Therefore, we choose $k = 5$ as the length of subsequences, sliding step size used is five.

In the earlier section, we mentioned the estimation of classifier generalization. The choice of the length of a subsequence impacts the bound on the actual risk of support vector machine. When it is assumed that the training set is a representative sample from true data distribution, then the number of support vectors is an indication of the expected error made on the target set. When the training set is a sample which only captures the area in the feature space, but does not follow the true probability density, it is expected that the error estimates is far off.

This estimate can be derived by applying leave-one-out estimation on the training set [42][53]. To estimate the leave-one-out error, we can distinguish three cases:

1. When one of the points (for which $\alpha_i = 0$) is left out during training and separating hyperplanes are computed, the same solution is obtained as with the training set including this training objects. During testing this object will therefore be assigned same label by the classifier.
2. When a support object (with $0 < \alpha_i < C$) is left out, this support point will be rejected or accepted by the new solution during testing.
3. When a support object (with $\alpha_i = C$) is left out of the training set, the original data description is still obtained. They will again be treated as an error during testing.

The error estimate then becomes:

$$\varepsilon \leq \frac{n_{SV}}{N} \tag{4.21}$$

n_{SV} is the number of support vectors. N is the number of training data.

Equation (4.13) can become:

Find α to minimize $W = \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_i \alpha_i + b\sum_i y_i \alpha_i$

Subject to $$0 \leq \alpha_i \leq C, \quad \forall i$$

Then the first-order KKT conditions may be stated:

$$\frac{\partial W}{\partial \alpha_i} = \sum_j \alpha_j y_i y_j K(x_i, x_j) + y_i b - 1 = y_i f(x_i) - 1 \begin{cases} > 0 & \alpha_i = 0 \\ = 0 & 0 < \alpha_i < C \\ < 0 & \alpha_i = C \end{cases}$$

With $\sum_i \alpha_i y_i = 0$ (4.22)

The derivation of the inequality function in the above is from the Equation (4.5). Figure 4.16 illustrates how SVM hyperplane partitions the training data and corresponding α_i in three categories. They are within the margin, on the margin and exceed the margin.

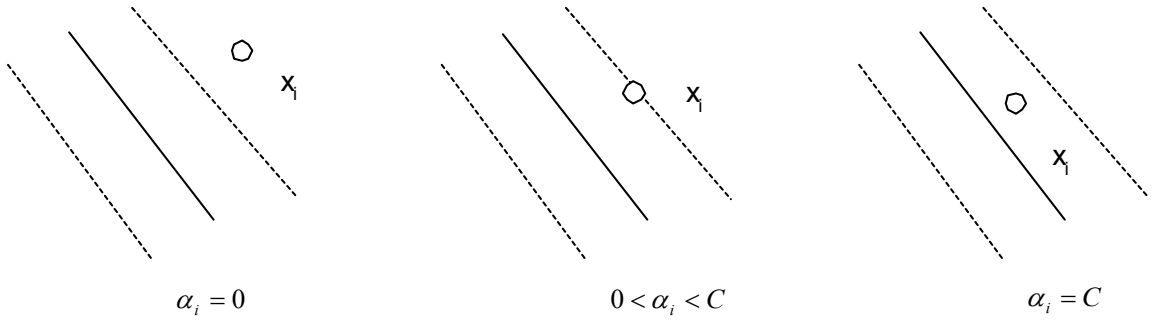


Figure 4.16 SVM Classifier

For two subsequence length k_1, k_2, based on the analysis of the subsequence length choice we did before, $K(x_i, x_j; k_2) < K(x_i, x_j; k_1) \quad \forall_{i,j}, \quad k_2 > k_1$. For larger k_2, $K(x_i, x_j; k_2)$ gets smaller. In terms of Equation (4.22), it indicates that more points become support vectors ($0 < \alpha_i \le C$) in order to minimize the Equation (4.13). By the use of formula (4.21), this also means an increase of the expected error on the target data. Table 4.2 compares the number of support vectors for different subsequence lengths. It is noticed that the number of support vectors increase with the increase of the subsequence length.

4.3.3 Kernel Function

One of the major tricks of SVM is the use of kernel function to extend the class of decision function to the non-linear case. This is done by mapping the data from the input space X into a high dimensional feature space H by a function $\Phi : X \to H$ and solving the linear learning problem in X.

After building the subsequence library from the training data, it is ready to define the feature mapping function $\Phi(S)$. The feature mapping function $\Phi(S)$ for any length string S is indexed by all possible subsequences of length k from the library [65].

Table 4.2 The Comparison of the Error Estimate with Respect to Subsequence Length

Faults	Subsequence Length	$\frac{n_{SV}}{N}$
Engine Faults	5	10.98%
	20	13.45%
Aileron Deflection	5	15.22%
	20	17%
Rudder Deflection	5	14.86%
	20	16.13%
Stabilizer Deflection	5	39.27%
	20	41.90%
Elevator Deflection	5	38.07%
	20	39.85%

We define the feature map from S to R^{n^k} by

$$\Phi(S) = (\phi_i(s))_{i \in L}\ ,\ i = 1,...,n \tag{4.23}$$

Where $\phi_i(s)$ is the number of times of *ith* subsequence in the library occurring in S. $\Phi(S)$ is a n-length vector (where n is the length of subsequence library).

In section 4.2, we describe that once there is defined mapping function $\Phi(S)$, we can take advantage of the kernel trick to define subsequence based kernel by calculating the inner product in the feature space. Therefore, the kernel value between two strings S_1, S_2 is defined as:

$$k(S_1, S_2) = \langle \Phi(S_1), \Phi(S_2) \rangle \tag{4.24}$$

Our subsequence based kernel is a sequence similarity kernel. To be specific, if two strings contain many of the same k-length subsequences, their inner product under the defined kernel in Equation (4.24) will be large. In this sense, the kernel $k(S_1, S_2)$ can measure the similarity of input strings S_1 and S_2. The more subsequences these two strings share, the bigger value the kernel has. The kernel will be close to zero for all non-matching subsequences. In section 4.3.2, we gave a test example to illustrate the similarity between strings measured by subsequence based kernel defined in Equation (4.24).

In this research, we use the normalized kernel that is defined as:

$$k_{norm}(S_i, S_j) = \frac{k(S_i, S_j)}{\sqrt{k(S_i, S_i) \cdot k(S_j, S_j)}} \tag{4.25}$$

4.3.4 Time Delay

One important issue in fault detection is the time delay problem. Our kernel function measures the similarity between two strings, rather than two symbols. Therefore, SVM makes the binary classification on a window of time series points. The length of this window is the time delay we point out here. The bigger the window, the longer the time delay for the fault detection. We name this sliding window the unit string. The unit string

has to be small in terms of the time delay problem. In the mean time it must contain enough information in order to enable the classification with SVM. After extensive experimental simulations, we choose the length of unit string as 100 since it compromises the time delay problem and classification accuracy problem. Details are referred to in Chapter 6.

4.3.5 Multiple SVM

In the above sections, we discussed how to train a SVM with only one sensor measurement. In fact, there are 12 states variables available. However, it is observed that not all the states obtained from the B747 model are indicative of faults. As in Chapter 3, only 7 state variables that are most indicative to faults are considered. Therefore, the state variables considered are true airspeed, angle of attack, slideslip angle, roll angle, pitch angle, yaw angle and altitude. It is required to train a SVM for every state variable. Then a final fault detection decision is made after performing the majority vote.

4.4 Online Fault Detection

Section 4.3 discussed how to develop a subsequence-based kernel with normal and faulty training data in the training phase. The next phase is the classification. The objective is to use the trained SVM hyperplane to classify new data into either a normal or an abnormal class. Figure 4.17 shows the fault detection procedure employing a majority voting mechanism on seven independent classifiers. The classification phases are the following:

- Convert the test data into a string x using Lloyd algorithm with 16 bins.
- Test each state of x with its corresponding trained SVM, that is, SVM for true airspeed (TA), SVM for angle of attack (AA), SVM for slideslip angle (SSA), SVM

for roll angle (RA), SVM for pitch angle (PA), SVM for yaw angle (YA) and SVM for altitude.

- Make the decision on every single SVM.
- Reach the final decision using a majority-voting scheme.

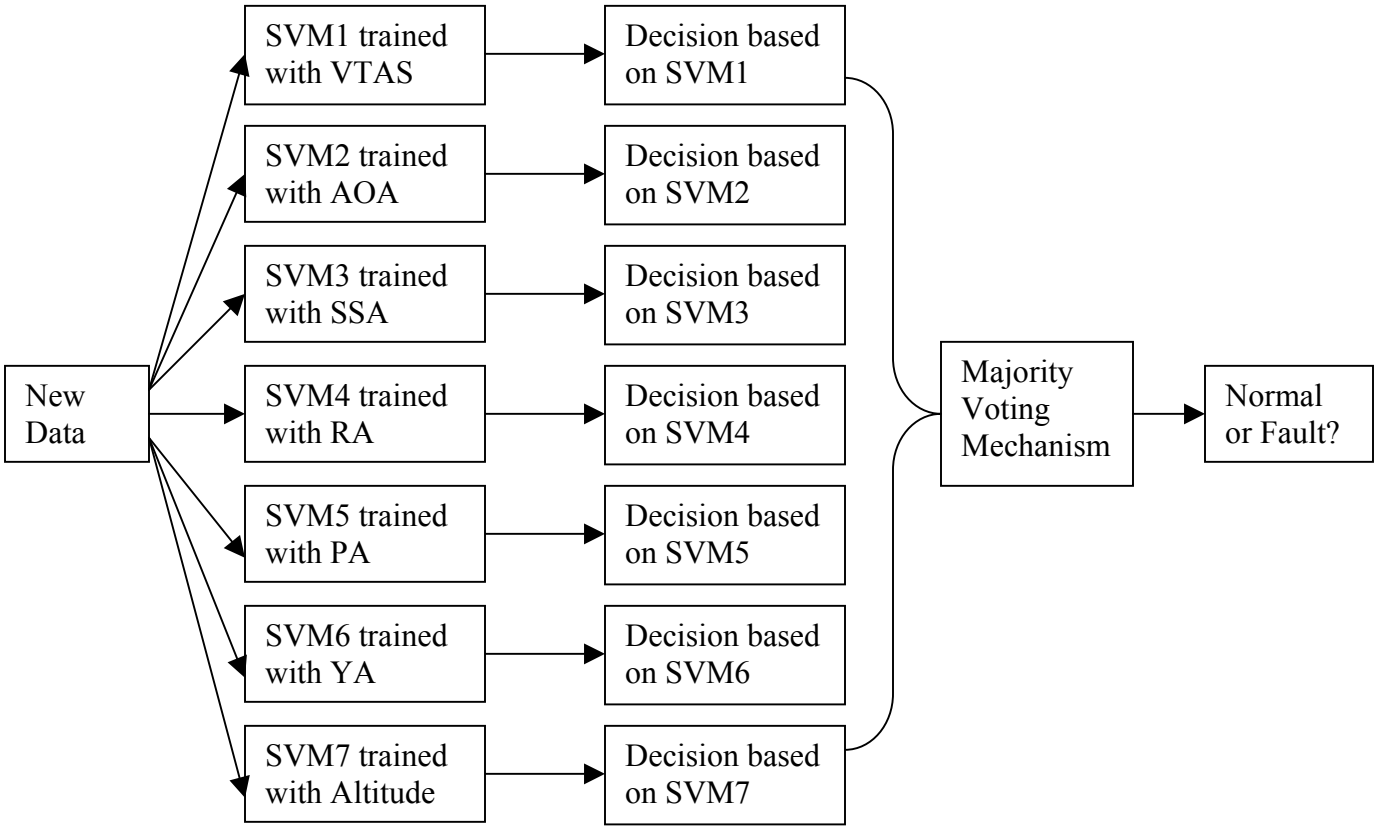


Figure 4.17 Majority Voting Schema

Chapter 5

Fault Isolation

In the last two chapters, we discussed how to determine if the new sensor data is normal or faulty with one-class and two-class classification. Once the system is declared faulty, the fault isolation should be followed, i.e. we need to determine the fault source. As mentioned earlier, we considered five faults: aileron, elevator, rudder, stabilizer and engine failure. The isolation task is to categorize a new sensor data into one of these faults.

5.1 Multi-Class Support Vector Machine

We use multi-class support vector machine in [66][67][68] with subsequence-based kernel developed in Chapter 4 for fault isolation. Support vector machines (SVM) were originally designed for binary classification. How to effectively extend it for multi-class classification is still an on-going research issue. Currently there are two types of approaches for multi-class SVM. One is by constructing and combining several binary classifiers while the other is by directly considering all data in one optimization formulation. The first type has two subclasses: one against all and one against one.

1. One against all

The earliest used implementation for SVM multi-class classification is probably the one against all method [66]. It constructs k binary SVM models for k fault classes. The ith SVM model is trained with all of the examples in the ith class with positive labels and all other examples with negative labels. Thus given l training data $(x_1, y_1),...,(x_i, y_i)$, where

$x_i \in R^n, i = 1,...,l$ and $y \in \{1,...,k\}$ is the class of x_i, the *ith* SVM solves the following problem [66]:

$$\min_{w^i, b^i, \zeta^i} \frac{1}{2}(w^i)^T w^i + C\sum_{j=1}^{l} \xi_j^i$$

$$\text{Subject to } (w^i)^T \phi(x_j) + b^i \geq 1 - \xi_j^i \text{ if } y_j = i$$

$$(w^i)^T \phi(x_j) + b^i \leq 1 - \xi_j^i \text{ if } y_j \neq i$$

$$\xi_j^i \geq 0, j = 1,...,l \tag{5.1}$$

After solving (5.1), there are k decision functions:

$$(w^1)^T \phi(x) + b^1$$
$$\ldots.$$
$$(w^2)^T \phi(x) + b^k$$

We can say x is in the fault class that has the largest value of the decision function:

$$\text{Fault class of } x = \arg\max_{i=1,...,k}((w^i)^T \phi(x) + b^i)$$

2. One against one

Another major method is called the one against one method. It was introduced in [67]. The method constructs $\frac{k(k-1)}{2}$ binary classifiers where each one is trained on data from two fault classes. For faulty training data from the *ith* and the *jth* fault classes, the following binary classification problem is solved [68]:

$$\min_{w^i, b^i, \zeta^i} \frac{1}{2}(w^{ij})^T w^{ij} + C\sum_{t=1}^{l} \xi_t^{ij}$$

$$\text{Subject to } (w^{ij})^T \phi(x_t) + b^{ij} \geq 1 - \xi_t^{ij} \text{ if } y_t = i$$

$$(w^{ij})^T \phi(x_t) + b^{ij} \leq 1 - \xi_t^{ij} \text{ if } y_t \neq i$$

$$\xi_t^{ij} \geq 0 \tag{5.2}$$

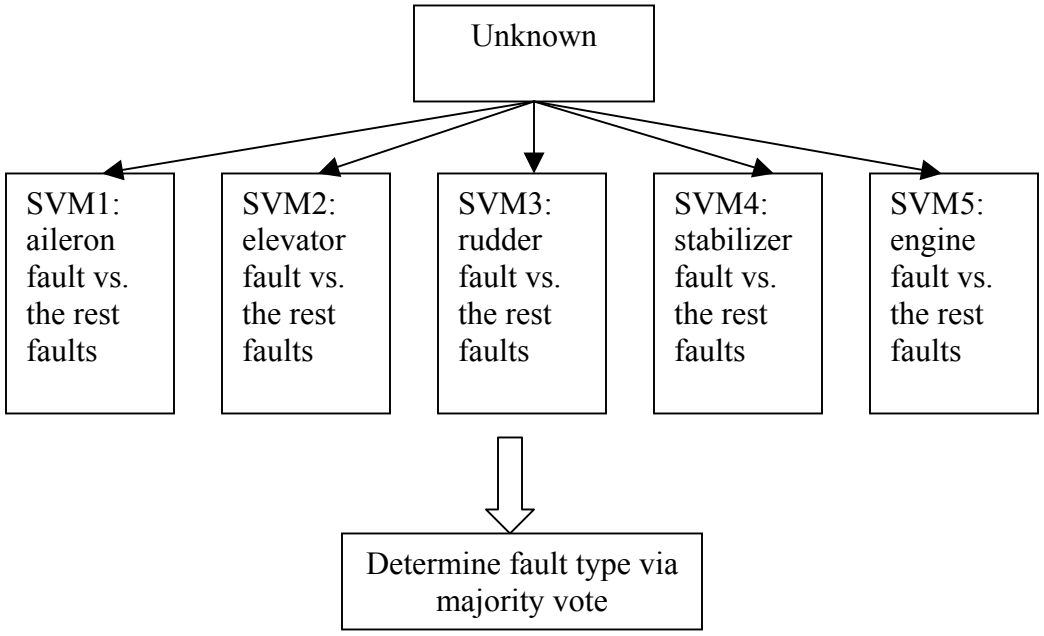


Figure 5.1 One Against All Fault Isolation Concept

There are different methods for doing the future testing after all $\frac{k(k-1)}{2}$ classifier are constructed. The voting strategy suggested in [69] is used for decision function: if $\text{sign}((w^{ij})^T \phi(x) + b^{ij}))$ says x is in the *ith* class, then the vote for the *ith* class is added by one. Otherwise, the *jth* is increased by one. Then we predict x is in the class with the largest vote.

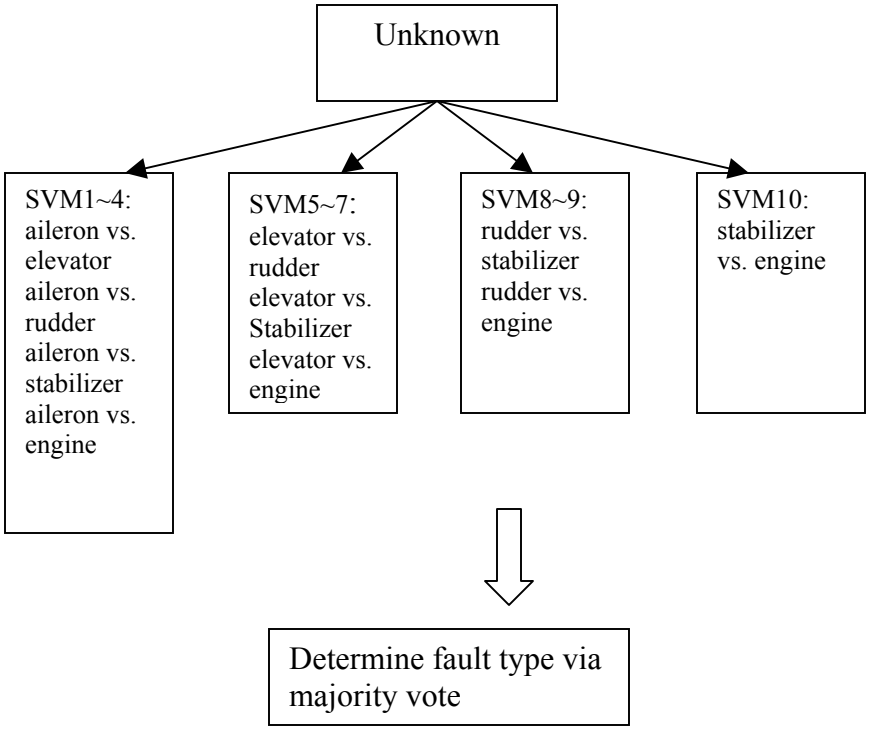


Figure 5.2 One Against One Fault Isolation Concept

3. Single optimization

The idea is similar to the one against all approach. It constructs k two-class rules where the mth function $w_m^T\phi(x)+b$ separates training vectors of the fault class m from the other vectors [68][70]. Hence there are k decision functions but all are obtained by solving one problem. The formulation is as follows:

$$\min_{w^i,b^i,\zeta^i} \frac{1}{2}\sum_{m=1}^{k}(w_m^T \cdot w_m) + C\sum_{j=1}^{l}\sum_{m\neq y_i}\xi_i^m$$

$$\text{Subject to } w_{y_i}^T\phi(x_i)+b_{y_i} \geq w_m^T\phi(x_i)+b_m+2-\xi_j^m$$

$$\xi_i^m \geq 0, i=1,...,l\ ,\ m\in\{1,...,k\}\setminus y_i \qquad (5.3)$$

Then the decision function is

$$\arg\max_{m=1,...,k}(w_m^T \cdot \phi(x)) + b_m),\ i \in \{1,...,k\}$$

Like binary SVM, it is easier to solve the dual problem here [70]. The dual formulation of (5.3) is

$$\begin{aligned}
&\min_{\alpha} \sum_{i,j}(\frac{1}{2}c_j^{y_i} A_i A_j - \sum_m \alpha_i^m \alpha_j^{y_i} + \frac{1}{2}\sum_m \alpha_i^m \alpha_j^m)K_{i,j} - 2\sum_{i,m}\alpha_i^m \\
&\sum_{i=1}^{l} \alpha_i^m = \sum_{i=1}^{l} c_i^m A_i, m = 1,...,k, \\
\text{Subject to} \quad &0 \le \alpha_i^m \le C, \alpha_i^{y_i} = 0, \qquad (5.4) \\
&A_i = \sum_{m=1}^{k} \alpha_i^m, c_j^{y_i} = \begin{cases} 1, y_i = y_j \\ 0, y_i \ne y_j \end{cases} \\
&i = 1,...,l, m = 1,...,k
\end{aligned}$$

Where $K_{i,j} = \phi(x_i)^T \phi(x_j)$.

Then

$$w_m = \sum_{i=1}^{l}(c_i^m A_i - \alpha_i^m)\phi(x_i), m = 1,...,k$$

And the decision function is

$$f(x,\alpha) = \arg\max_{m=1,...,k}(\sum_{i=1}^{l}(c_i^m A_i - \alpha_i^m)K(x_i, x) + b_m)$$

We compare these three methods mentioned above on the same simulation data. In the training phase, we build multi-class SVMs with fault training data. In the test phase, for a particular test string, we measure the similarity to each defined fault class by calculating the subsequence-based string kernel value between this test string and every training string with known fault class. We use same seven kinds of sensor measurements as the fault detection does. The final decision of fault class is made if the majority of the sensors declare it to be a particular fault. The computation of similarity is performed in the same way as the one in the fault detection part and hence is not repeated here.

Chapter 6

Simulations

6.1 One-Class Classification

Our test bed is derived from NASA's model for a B747 aircraft originally implemented in the DASMAT environment by Gary Balas of the University of Minnesota [71]. The data for analysis was obtained by using the linear model of B747 as an independent Simulink model. Six different faults are implemented. They include two engine failures, elevator deflection, rudder deflection, aileron deflection, and stabilizer deflection faults. The model is implemented in open loop and also under a flight controller designed as a linear quadratic regulator. In open loop simulation, all the actuators are given zero mean independent random variations (Figure 6.1).

For the cases shown here the variations are of a maximum magnitude equals to 1% of the range that particular actuator. The duration of simulation is 100 seconds, the sampling time of 0.5 seconds. The time at which the fault is introduced occurred at 50 seconds through 70 seconds. Engine failure faults are generated by giving thrust loss of certain amount of Newton. Other faults are produced by giving the control surfaces a step change. The intensity of the fault is the magnitude of the step measured as percentage of the actuator range. About the sensitive variables we use are angle of attack, roll angle, pitch angle and yaw angle. Angle of attack and pitch angle are more sensitive in stabilizer and elevator deflection. Roll angle and yaw angle are sensitive in the rest of other faults. Under regular flight control, the manipulated variables should experience small fluctuations to compensate for “normal disturbances” while the controlled variables should remain, ideally, at their trimming values. Therefore, it is also necessary to

consider this situation in establishing the characteristics of the normal set. For the case discussed here we assume no measurement of controlled variables and instead of collecting data to get an estimate of a normal convex set, we use a computer-based simulator to gather data. We illustrate this issue by considering the aircraft operating in open loop and giving the inputs small changes in the neighborhood of the trimming controls. The measured variables are the twelve state variables of a six-degrees of freedom model, namely: roll rate, pitch rate, yaw rate, true airspeed, angle of attack, sideslip angle, roll angle, pitch angle, yaw angle, altitude, x-position (forward), y-position (lateral). The four control surfaces and their ranges in degrees are: stabilizer deflection [-12~+3], aileron deflection [-20~+20], rudder deflection [-25~+25], elevator deflection [-23~17]. Each of the four engines thrust has the range [0~44000] N.

The following results illustrate the tradeoff between size of normal set and detection capability. The normal set is determined by flying the plane and giving to each control inputs a zero mean random variation. For the cases shown here, there were ten different normal flights, and 200 measurements were collected in each flight at two measurements per second. The measure of fault severity is the fraction of the actuator range that the fault causes and the variation is the fraction of the range used in determining the normal convex set. Thus a variation of 0.01 means that the normal fluctuations are 1% of the normal range. For example, Figure 6.2 shows 20 roll angle normal data sets with 1% normal fluctuation. We implement actuator faults by introducing piecewise constant changes in the corresponding input. In the following simulation, we give the 1% normal variation. Fault severity is given within [0.04 0.1] and different random seeds are chosen to create faulty data.

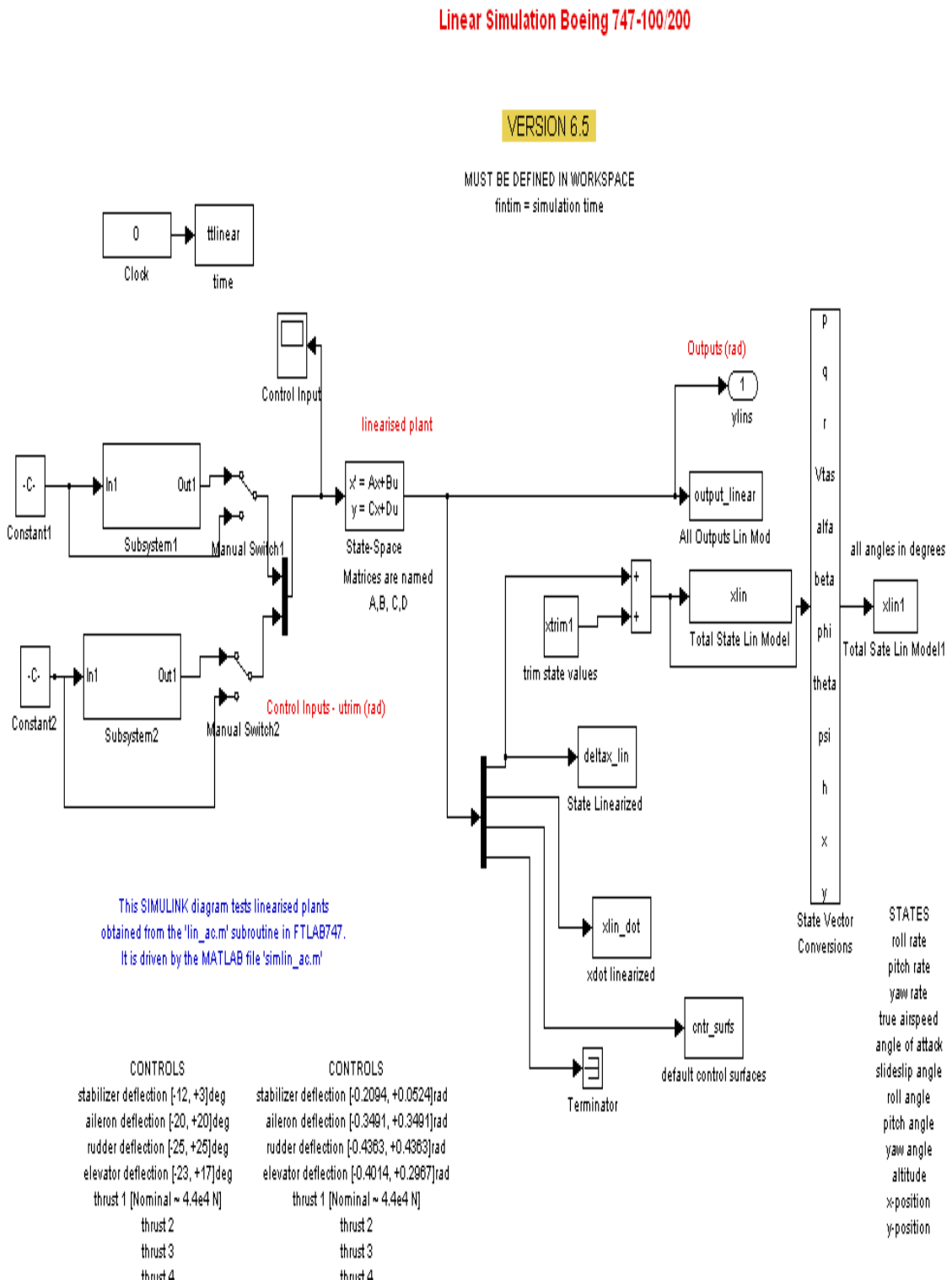


Figure 6.1 Open Loop Linear Simulink Model

The results show that all faults, except stabilizer, can be detected. The false alarm rate is zero. There is a detection time delay and it is inversely proportional to the fault severity. Figure 6.3 shows this change. The results of the simulations are very consistent and intuitive. For the limitation of space, we only show the fault severity of 0.1 and 0.04, normal variation of 0.01 and 0.02. From Table 6.1, one can see missed alarm will increase with the increase of normal variation and less fault severity. Large faults and smaller normal variation are detected quickly and some variables are more sensitive to certain types of faults.

- **Trend Indicator -- Distance to normal convex hull**

We can visually show how test flight data deviate from the normal convex hull. The test flight data is engine failure fault. The duration of simulation is 100 seconds, the sampling time of 0.5 seconds. The time at which the fault is introduced occurred at 50 seconds through 70 seconds. The normal convex hull contains 144 facets. The distance vector is calculated by each test data from every single facet of normal convex hull. Figure 6.4 gives these distance trends. It is noticed that before 100^{th} point in the X-axis the distance is negative. This indicates that the test data points are within the normal convex hull, and they are normal data. Right after that, the distance vectors slowly move to positive direction. It proves that the test data points are moving outside of normal convex hull, becoming faulty data. In order to do fault detection, we calculate the standard deviation of those distance vectors (Figure 6.5). The reason why we choose standard deviation is that the distance variation within normal convex hull is quite smaller than the one outside of normal convex hull. In order to detect the trend, we simply use the difference between the current value and its previous one (Figure 6.6).

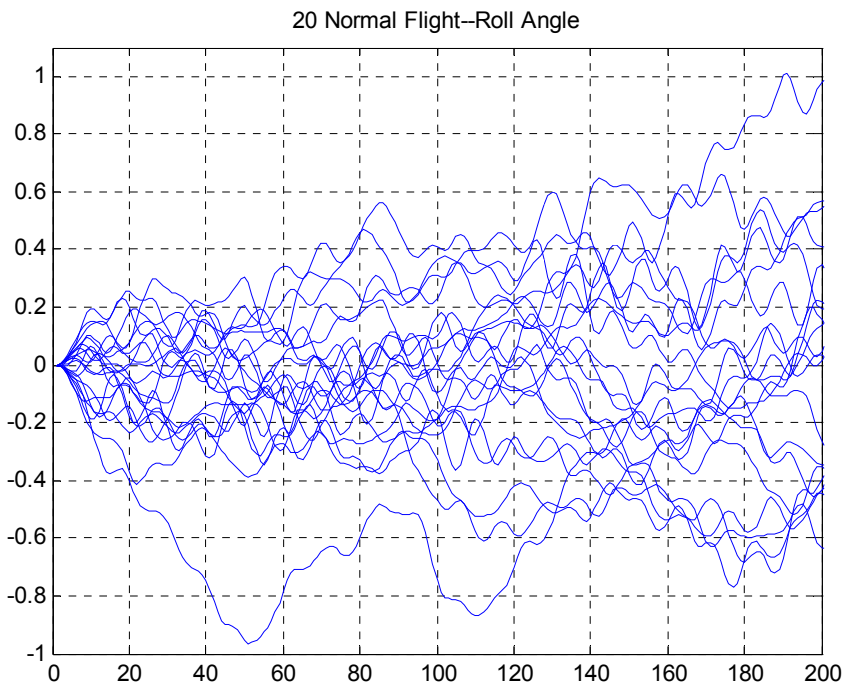


Figure 6.2 Normal Class

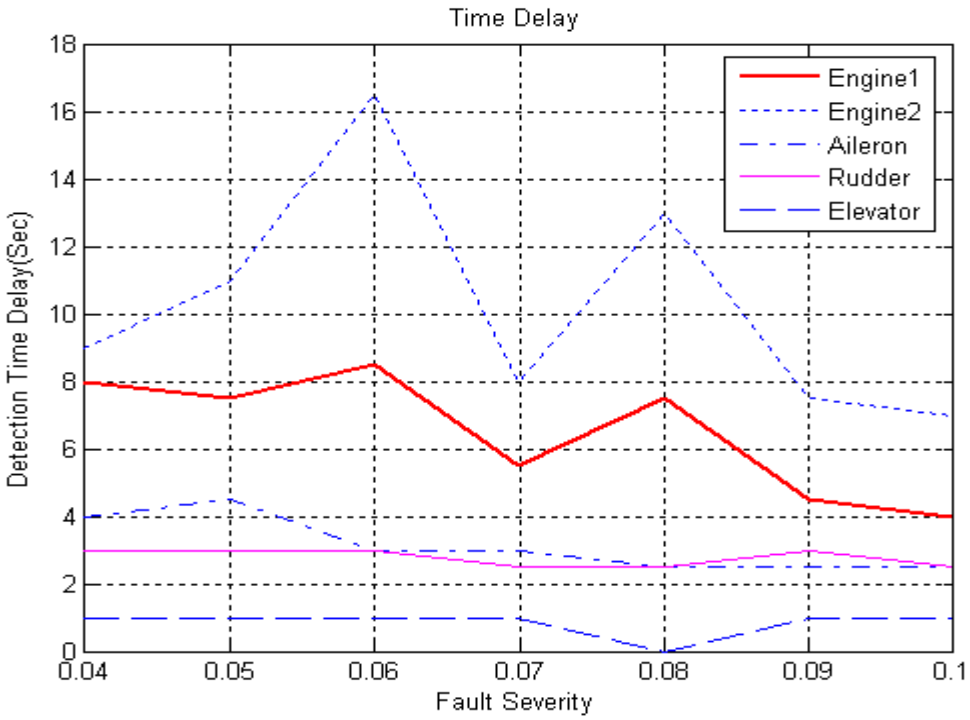


Figure 6.3 Time Delay Vs Fault Severity For Five Faults

Table 6.1 Fault Detection Result for Different Fault Severity and Normal Data Variation

Severity=0.1, Variation=0.01

Faults	False alarm (%)	Missed alarm (%)	Time delay (s)
Stabilizer	0	50.25	
Aileron	0	0	1.5
Rudder	0	0	2
Elevator	0	1.49	1
Engine 1	0	0	3.5
Engine 2	0	0	7.5

Severity=0.04, Variation=0.01

Faults	False alarm (%)	Missed alarm (%)	Time delay (s)
Stabilizer	0	50.25	
Aileron	0	0	3.5
Rudder	0	0	3
Elevator	0	4.48	1
Engine1	0	0	7.5
Engine2	0	0	9

Severity=0.1, Variation=0.02

Faults	False alarm (%)	Missed alarm (%)	Time delay (s)
Stabilizer	0	50.25	
Aileron	0	0	3
Rudder	0	0	2.5
Elevator	0	2.99	1
Engine 1	0	0	7
Engine 2	0	0	9

Severity=0.04, Variation=0.02

Faults	False alarm (%)	Missed alarm (%)	Time delay (s)
Stabilizer	0	50.25	
Aileron	0	0	4.5
Rudder	0	0	3.5
Elevator	0	13.43	1
Engine 1	0	8.96	10
Engine 2	0	26.87	20.5

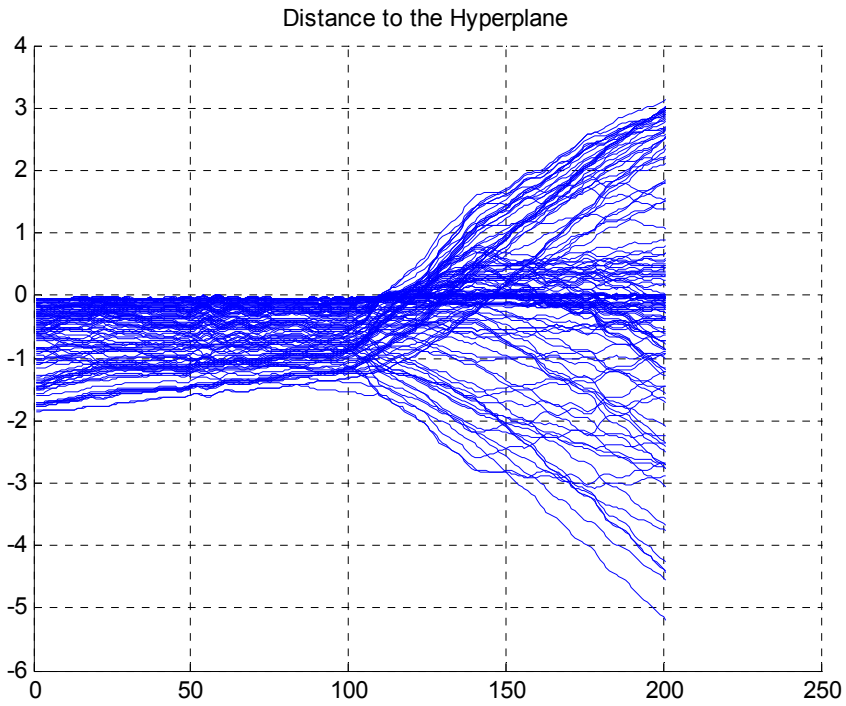


Figure 6.4 Distance of Test Data to Normal Convex Hull

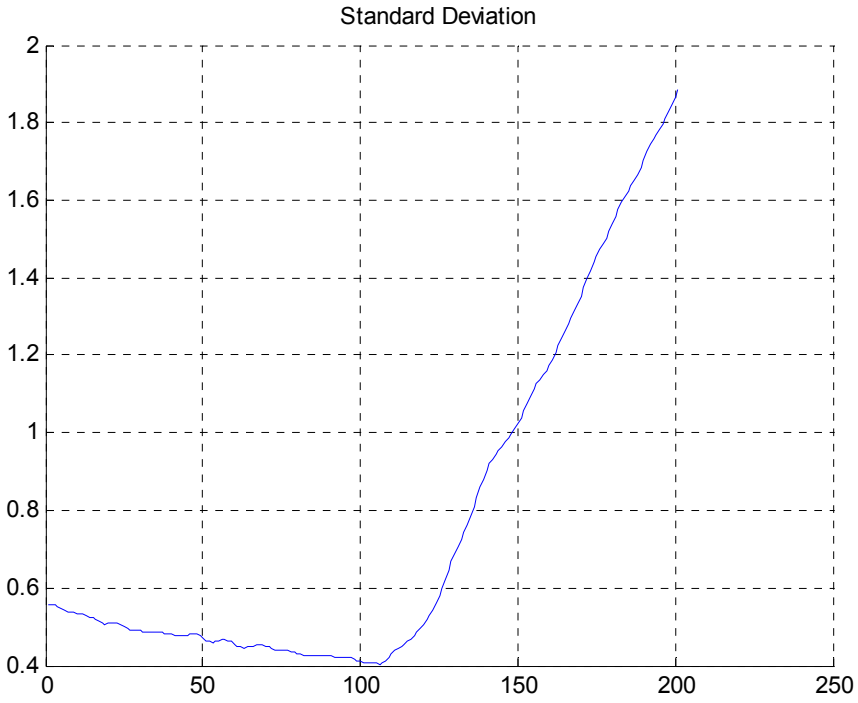


Figure 6.5 Standard Deviation of Distance

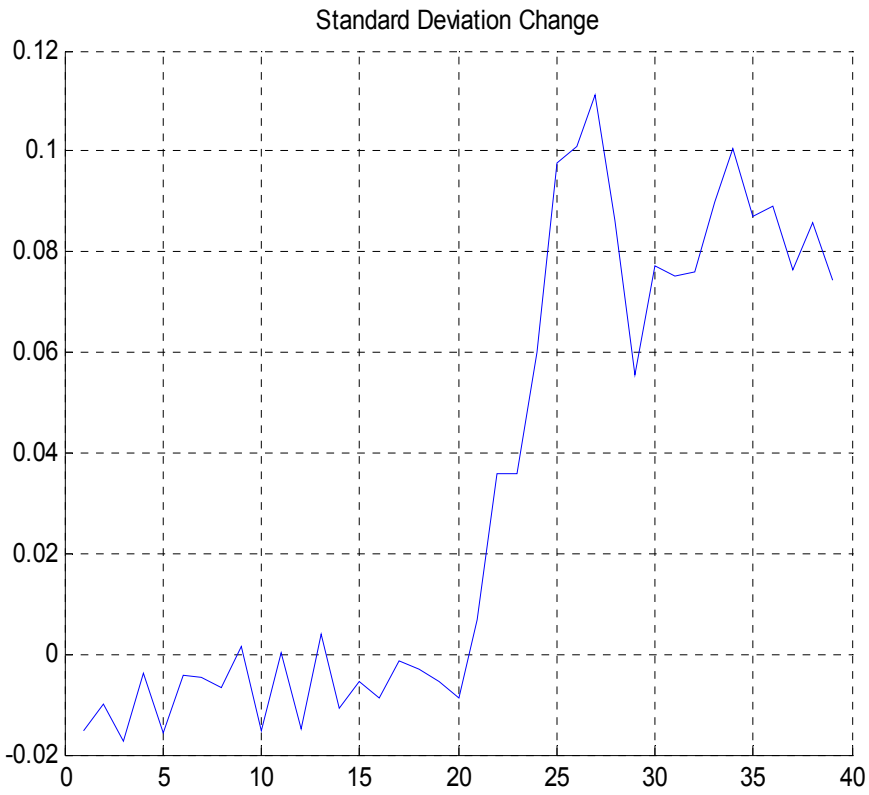


Figure 6.6 Trend Indicator – Difference of Standard Deviation of Distance

6.2 Two-Class Classification

We first test two-class classification with simulation in Section 6.1 to compare with one-class classification. In Table 6.2, it is noticed that time delay for two-class classification is smaller than one-class classification; however the accuracy of detection is a little lower than one-class classification. One thing needs to mention is the computation time is much shorter than one-class classification. It enables that the two-class classification more powerful for real-time fault detection.

Table 6.2 Fault Detection Result for Different Fault Severity and Normal Data Variation

Severity=0.1, Variation=0.01

Faults	False alarm (%)	Missed alarm (%)	Time delay (s)
Stabilizer	10.4	31.33	10
Aileron	0	0.33	0
Rudder	0	0	0
Elevator	2.16	1.49	0.5
Engine 1	0	0	0.5
Engine 2	0	0	3.5

Severity=0.04, Variation=0.01

Faults	False alarm (%)	Missed alarm (%)	Time delay (s)
Stabilizer	26.87	30.25	10
Aileron	1.22	2.46	0.5
Rudder	0	0	0.5
Elevator	1.54	3.69	1
Engine1	1.04	1.88	1.5
Engine2	1.69	1.15	3.5

Severity=0.1, Variation=0.02

Faults	False alarm (%)	Missed alarm (%)	Time delay (s)
Stabilizer	10.89	33.17	10.5
Aileron	0.33	0.33	0.5
Rudder	0	0	1
Elevator	2.33	3.46	0.5
Engine 1	0.66	1.18	2
Engine 2	0	0.33	3

Severity=0.04, Variation=0.02

Faults	False alarm (%)	Missed alarm (%)	Time delay (s)
Stabilizer	0	50.25	
Aileron	1.24	2.46	1
Rudder	0.33	0	1
Elevator	1.67	4.34	2
Engine 1	1.88	2.12	4
Engine 2	1.54	10.23	5.5

In the following we compare the performance of one-class classification and two class classification with closed loop non-linear testbed simulation. The test bed for two-class classification is still derived from NASA's model for a B747 aircraft. The data for analysis was obtained by using the nonlinear model of B747 as an independent Simulink model. This comprehensive closed loop non-linear testbed model has been developed using the software package FTLAB747. Different flight trimming conditions (straight and level, level turn, etc.) can be implemented using the model. This research uses the straight and level trim and develops a methodology to detect failure in elevator, aileron, rudder, stabilizer and engine. This systematic procedure can be applied to detect any fault for any flight condition. The faults for a particular actuator can be varied by changing two parameters, namely: the intensity of fault and the time of onset of fault. Inputs to the system are the initial conditions that help to repeat the exact input for different experiments. Also duration of simulation and the integration step can be varied in the Simulink block model that runs the simulation.

The simulations are carried out using the high fidelity model for the Boeing 747-100/200 previously trimmed at an equilibrium point. The aircraft configuration for this equilibrium point is as follows: The selected aircraft mass is 300,000 kg, and the position of the aircraft's center of gravity with respect to the (x, y, z)-axes is assumed to be 25 percent of the mean aerodynamic chord (m.a.c) for the x-axis and the point (0,0) meters for the other two axes. For normal operation of the flight, no fault is assumed, and a flight condition defined to be straight-level flight at 7000 meters of altitude and at a true airspeed of 218 m/sec is given. The Simulink diagram of the closed-loop nonlinear model is shown in Figure 6.7.

The flight is simulated using the closed-loop nonlinear model for a simulation time of 60 seconds with an integration step of 0.02 seconds. For simplicity, we have not introduced any noise in the sensors or the actuators. The fault generator block allows simulating a variety of faults, such as failures in actuators (elevator, aileron, rudder, stabilizer and engine). The sensor data available from the above model are the control inputs and the state variables. The control inputs include four control surfaces plus four engines. The ten state variables of the nonlinear model are: roll rate, pitch rate, yaw rate, virtual true airspeed, angle of attack, slide slip angle, roll angle, pitch angle, yaw angle, altitude. In this research we chose the state variables as the sensor measurements ignoring the measurements from the actuators. The methodology allows new data if desired. Out of ten sensors the first three variables i.e. the roll rate, pitch rate and yaw rate are just the derivation of their respective angles with respect to time. In this research we pick seven of them for multivariate time series fault detection. They are true airspeed, angle of attack, sideslip angle, roll angle, pitch angle, yaw angle, altitude. Each state variable is individually used to construct string kernel for fault detection. The final classification decision is made by the majority-voting scheme.

For each actuator we introduce a loss of efficiency fault between [0.02 0.2] to get a set of faulty data for each actuator. Fault onset time is at 0 second. For simulations performed for these tests, the plane is disturbed by adding a change to the initial state and then allowing the control system to return it to steady state. All the experiments have the same initial disturbances for comparison.

We use sequential minimal optimization (SMO), a SVM software implementation which implements the soft margin optimization algorithm described in [72].

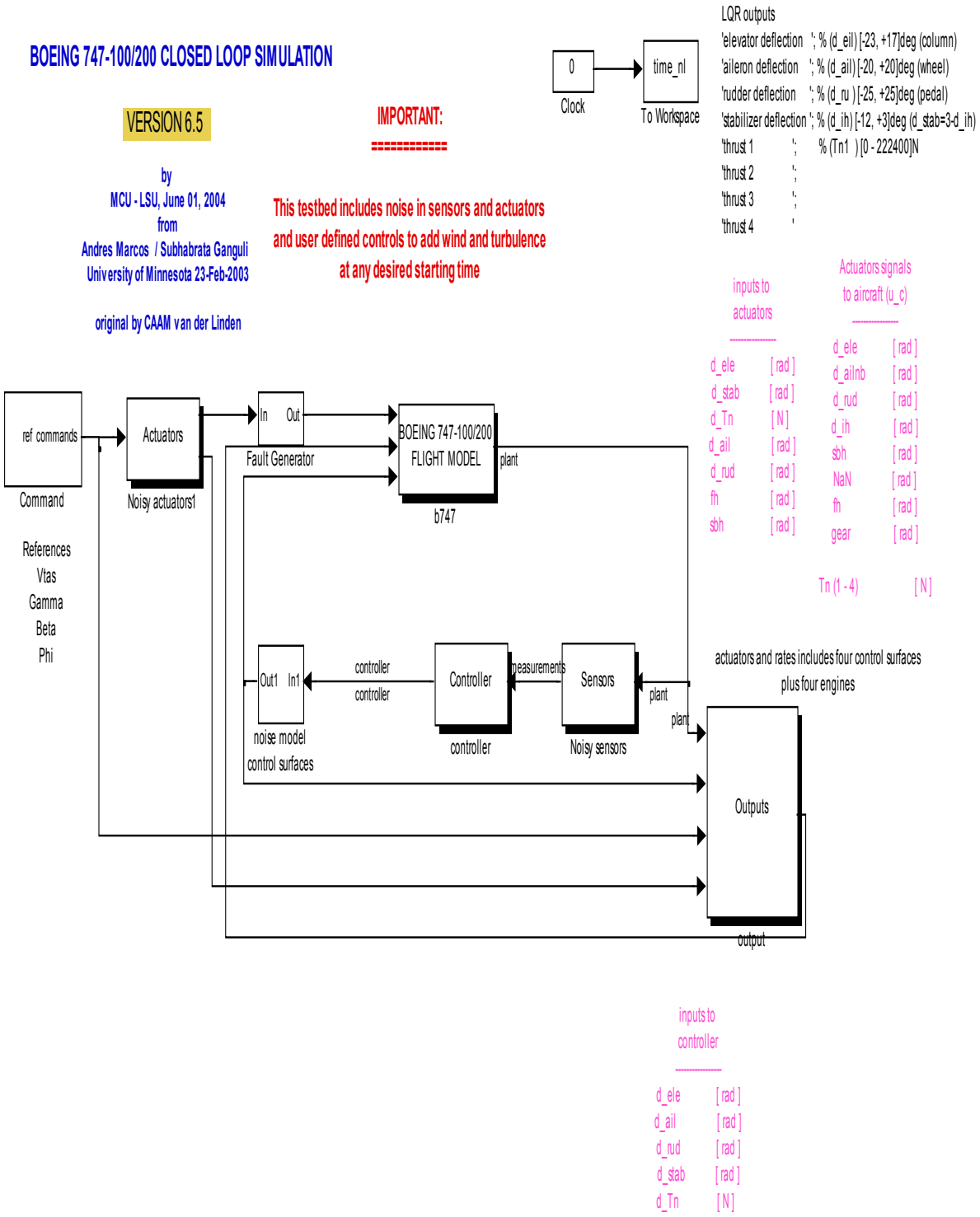


Figure 6.7 Closed Loop Non-Linear Simulink Model

Table 6.3 shows the mean false alarm rate and mean missed alarm rate from 70 simulations (each has 3000 measurements) for each fault with a varied combination of bins number and subsequence length.

Table 6.3 Comparison of Different Bins Number and Subsequence Length

Faults	Bins Number	Subsequence Length	False Alarm Rate	Missed Alarm Rate	Time Delay (sec)
Engine Failure	16	5	8.33%	0.33%	2.02
		15	9%	0%	2.02
	32	5	10.23%	0.14%	2.02
		15	11.67%	0.14%	2.02
Aileron Deflection	16	5	9.33%	1.67%	2.04
		15	10%	1.67%	2.04
	32	5	11.33%	1.33%	2.02
		15	12.23%	1.33%	2.04
Rudder Deflection	16	5	12.33%	0.17%	2.02
		15	14.67%	0.33%	2.02
	32	5	15.54%	0.17%	2.02
		15	16.83%	0.14%	2.02
Stabilizer Deflection	16	5	27.67%	7.67%	2.02
		15	25.67%	7.67%	2.02
	32	5	28.33%	6.83%	2.02
		15	30.34%	6.83%	2.02
Elevator Deflection	16	5	25.67%	9.58%	2.02
		15	26.67%	10.33%	2.02
	32	5	27.22%	10.04%	2.02
		15	29.54%	11.54%	2.02

There are several conclusions we can make from Table 6.3.

1. The shorter subsequence can gives higher accuracy on fault detection.
2. Less bins number provides lower false alarm rate than larger bins number. In terms of missed alarm rate, larger bins number choice can present a little bit lower missed alarm rates.

3. The computation time for the combination of larger bins number and longer subsequence length is much longer than smaller bins number and shorter subsequence length.

4. The most difficult faults to detect are the stabilizer failure and elevator failure.

For comparison, we use one-class classification to test same nonlinear closed loop simulation data (Table 6.4). Obviously, the detection accuracy is higher than two-class classification except the stabilizer fault. However, the computation time is much longer than two-class classification. It is not appropriate for real-time fault detection use. Therefore, if the faulty training data are available, we recommend applying two-class classification for fault detection.

Table 6.4 Closed Loop Simulation Fault Detection Result with One-Class Classification

Faults	False alarm (%)	Missed alarm (%)	Time delay (s)
Stabilizer Failure	0	50.25	
Aileron Failure	0	0	0
Rudder Failure	0	0	0
Elevator Failure	0	0	0
Engine Failure	0	0	0

Although we take each experiment with 3000 time points as one string, we need to look for the shortest string that can be classified as normal or faulty one. After testing different size of time points (50~150, corresponding to 1~3 seconds), we found the

minimum optimum string length for all kinds of faults is 100 for $k = 5$. Figure 6.8~Figure 6.12 shows the missed alarm rate and false alarm rate of the elevator failure, aileron failure, rudder failure, engine failure and stabilizer failure. It is noticed that the missed alarm rate drops significantly at the length of 100.

6.3 Fault Isolation

For all the data used above we also performed fault isolation strategy described in Chapter 5. Isolated five different faults are aileron failure, engine failure, rudder failure, stabilizer failure and elevator failure. Figure 6.13 shows the histogram of these five different faults. If their histograms are different it implies that these faults can be separable.

The results of 100 simulations (20 simulations for each fault) are summarized in the next table. Table 6.5 compares the fault isolation accuracy rate with three different multi-class SVM. It is noticed that the one against one method outperforms the other two. Table 6.6 further shows that not only the percentage of wrong detection but also what fault class was determined in one against one method. Thus for example, in the case of faults in the aileron we have correctly identified the event in 80.83%, 8.58% was wrongly identified as engine fault, 7.59% was wrongly identified as rudder, 1.3% was wrongly identified as elevator fault, 1.7% was wrongly identified as stabilizer fault. Elevator fault and stabilizer fault are hard to be isolated compared to the other three faults. The rudder fault is the easiest to be isolated among all faults.

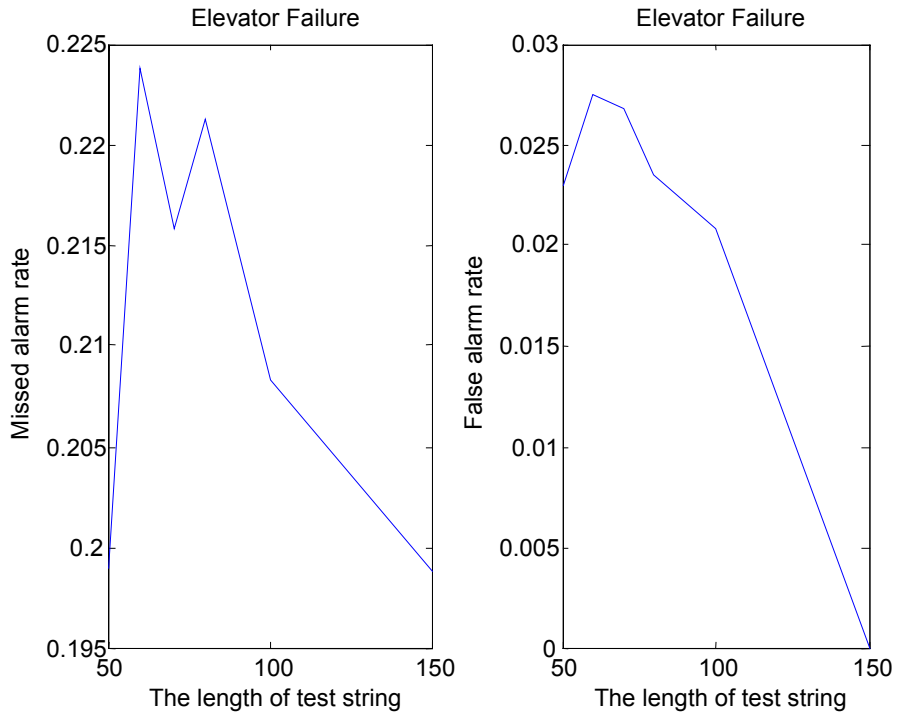


Figure 6.8 Elevator Failure

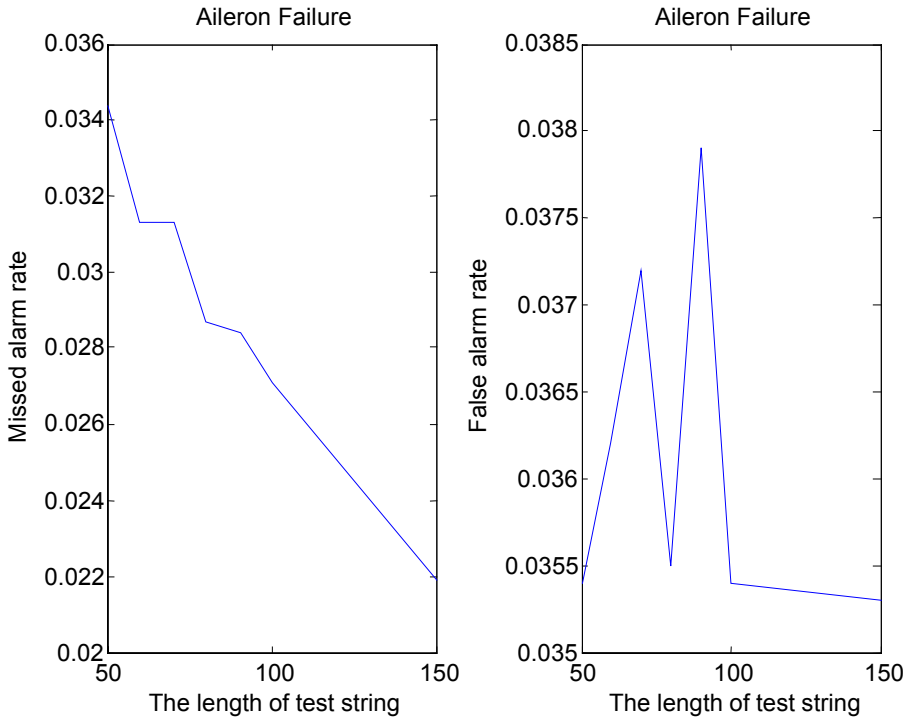


Figure 6.9 Aileron Failure

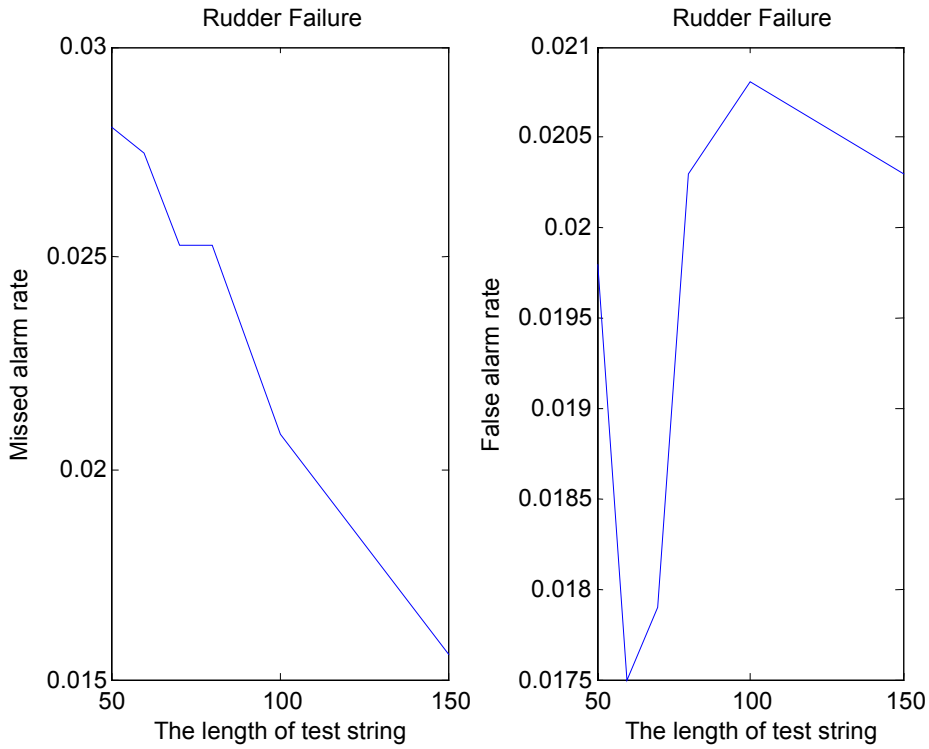


Figure 6.10 Rudder Failure

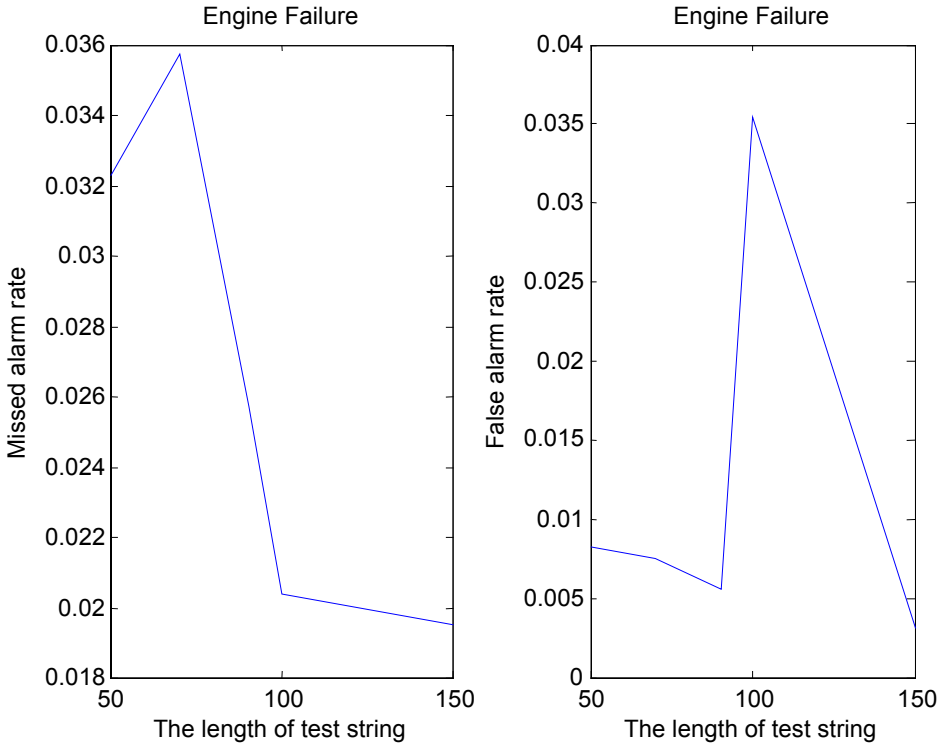


Figure 6.11 Engine Failure

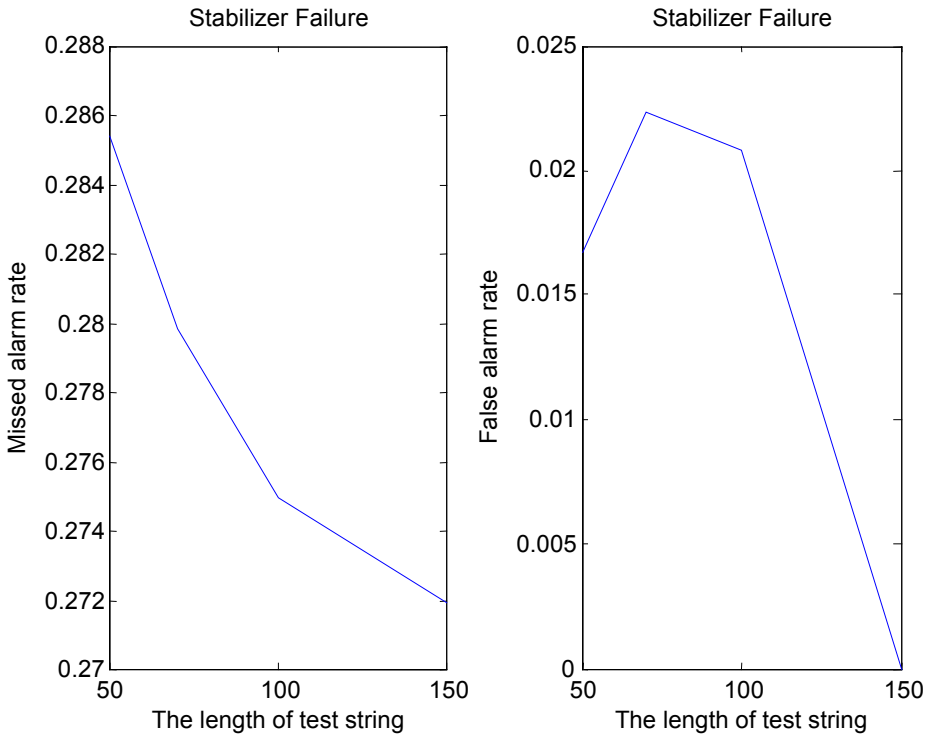


Figure 6.12 Stabilizer Failure

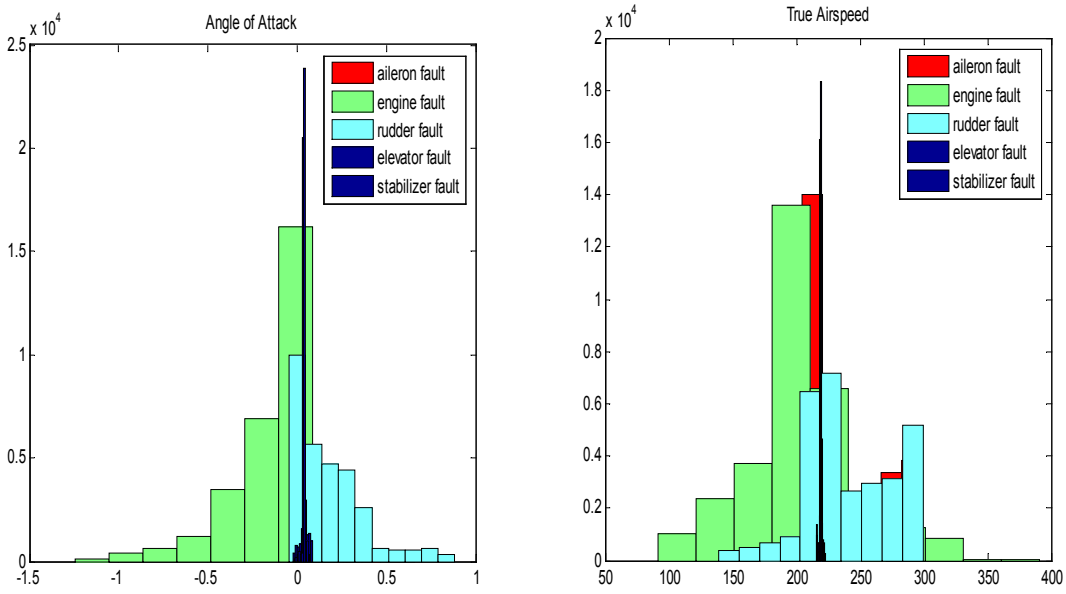
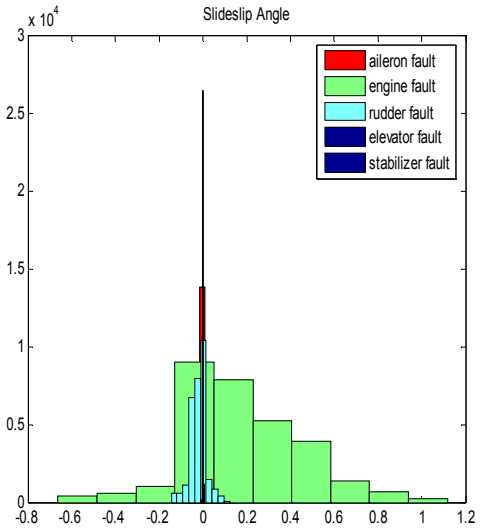
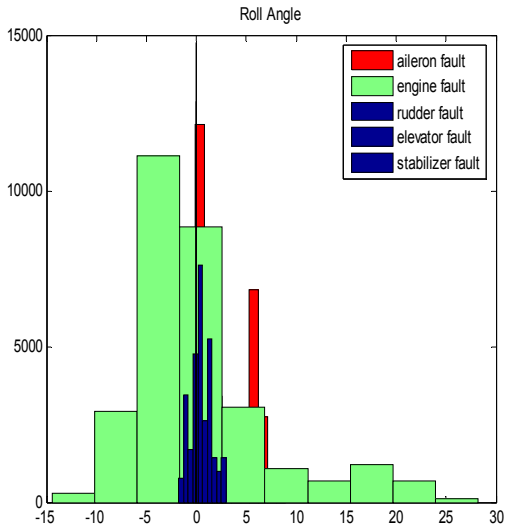
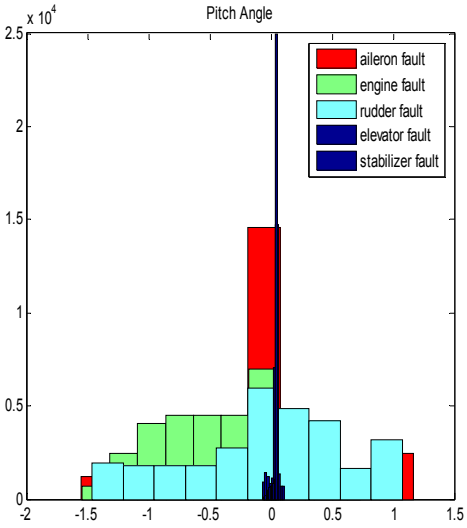


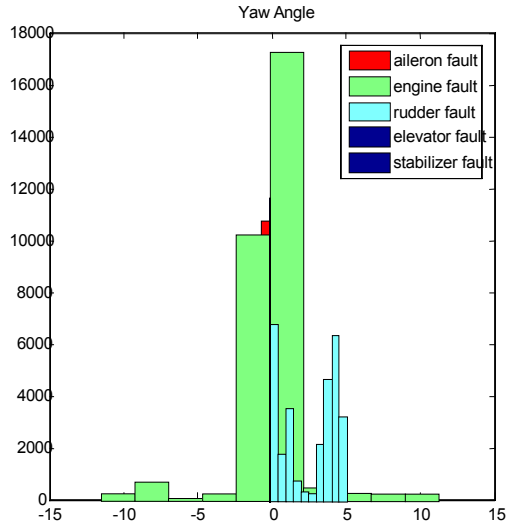
Figure 6.13 Histogram of Different Faults

(Figure Continues.)


Slideslip Angle
x 10^4
aileron fault
engine fault
rudder fault
elevator fault
stabilizer fault


Roll Angle
aileron fault
engine fault
rudder fault
elevator fault
stabilizer fault


Pitch Angle
x 10^4
aileron fault
engine fault
rudder fault
elevator fault
stabilizer fault


Yaw Angle
aileron fault
engine fault
rudder fault
elevator fault
stabilizer fault

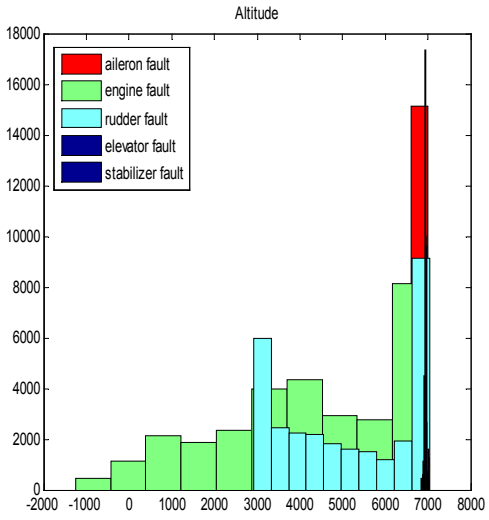

Altitude
aileron fault
engine fault
rudder fault
elevator fault
stabilizer fault

Table 6.5 Fault Isolation Performance with Different Multi-Class SVM Approach

	One against All	One against One	Single Optimization
Aileron Failure	64.57%	80.83%	72.08%
Engine Failure	71.25%	84.17%	85.83%
Rudder Failure	97.08%	92.5%	92.92%
Elevator Failure	50.53%	54.37%	55.39%
Stabilizer Failure	44.67%	45.89%	47.16%

Table 6.6 Fault Isolation with One against One Method

Actual Fault Class	Isolated Fault Classes				
	Aileron Failure	Engine Failure	Rudder Failure	Elevator Failure	Stabilizer Failure
Aileron Failure	80.83%	8.58%	7.59%	1.3%	1.7%
Engine Failure	6.92%	84.17%	6.91%	1.02%	0.98%
Rudder Failure	3.75%	3.75%	92.5%	0	0
Elevator Failure	4.6%	0	1.02%	54.37%	40.01%
Stabilizer Failure	1.06%	0.98%	1.31%	50.76%	45.89%

Chapter 7

Conclusion and Future Work

This research studies the data driven fault detection methods which rely on extensive collection of data instead of mathematical models to establish classification schemas that detect faults. Two different trending approaches are developed. One uses the normal data to define a one-class classifier. The second approach uses support vector machine (SVM) to define multi-class classifiers. In one-class classification, only normal class data are used to determine the closed boundary of the normal class. It is assumed that the normal operation sensor data should cluster in a convex set centered at the trimming values. Therefore we use the convex hull of the training set of points to describe the boundary of the normal class. By determining whether the new sensor data is inside the convex hull we can conclude if it is normal sensor data. Moreover, the distance from the test data to the normal convex hull is a trend indicator of test flight data safety status. We also discuss how to recursively adapt the boundary when the new incoming normal data deviates from the training data distribution.

When enough representative faulty data are available, we can create the binary classifier. The binary classifier SVM is able to separate normal data from faulty one. Subsequence based kernel function is proposed to measure the similarity of input strings. We first map the real-valued data into the discrete form via a non-uniform quantization based on Lloyd's algorithm. An analog value is normalized with respect to a defined range and discretized into bins. Each data is assigned a symbol corresponding to the bin in which it falls. Thus, the continuous numeric data have been transformed into the discrete character string. Then we build a subsequence library recording all possible k-

length contiguous subsequences from normal and faulty training strings. The feature mapping function $\Phi(S)$ for any length string S is defined as an n-length vector. Each element is the number of times of the k-length subsequence in the library occurring in S. The more subsequences two strings share, the bigger value the kernel is. Furthermore, multi class classifier is implemented through several binary SVM classifiers to isolate different faults.

The test bed used to collect data and implement the fault detection is a six degrees of freedom, rigid body model of a B747 100/200 and only faults in the actuators are considered. The simulation results prove that one-class convex hull and two-class SVM are promising tools for data-driven fault detection. They both have very short time delay, low false alarm rate and missed alarm rate. However, for the one-class classifier the computation time gets longer when more sensor variables are used. Future work can be done to improve the computation efficiency. Because only the normal class of data is available in one-class classification, it is hard to decide how tightly the boundary should fit around the data. Future focus can be on how to estimate the error of the faulty class.

In the binary classifier SVM, we consider each measured state variable independently. However, in the future it is worthwhile to study the correlation between state variables in order to explore the possibility of multi-dimensional string vector. Further comparison of SVM with some other traditional data-driven approaches can be helpful. In the majority-voting scheme, it is better to assign the different weight to each vote from the individual SVM based on the significance level of different faults.

References

[1] R. Isermann and P. Balle. Trends in the application of model-based fault detection and diagnosis of technical process. *Control Engineering Practice* 5(5), pp. 709-719, 1997.

[2] *The NASA Aeronautics Blueprint-Toward a Bold New Era of Aviation*, 2002.

[3] Y. M. Zhang and X. R. Li. Detection and diagnosis of sensor and actuator failures using IMM estimator. *AES-34(4)*, pp.1293-1312, Oct, 1998.

[4] A.S. Willsky. A survey of design methods for failure detection in dynamic systems. *Automation*(12), pp.601-611, 1976.

[5] J. Chen and R.J. Patton. *Robust Model-based Fault Diagnosis for Dynamic Systems*. Boston, MA, Kluwer, 1999.

[6] P.M. Frank. Analytical and qualitative model-based fault diagnosis – a survey and some new results, *European Journal of Control* 2(1), pp. 6-28, 1996.

[7] Venkat Venkatasubramanian and Raghunathan Rengaswamy. A review of process fault detection and diagnosis Part III: Process history based methods. *Computers and Chemical Engineering* 27, pp. 327-346, 2003.

[8] H. Vedam and Venkatasubramanian, V. A wavelet theory-based adaptive trend analysis system for process monitoring and diagnosis. *American Control Conference* pp. 309-313, 1997.

[9] K. B. Konstantinov and T. Yoshida. Real-time qualitative analysis of the temporal shapes of the (bio) process variables. *American Institute of Chemical Engineers Journal* 38 (11), pp.1703-1715, 1995.

[10] Jorge Aravena and F. Chowdhury. Fault detection of flight critical systems. *Proc. 20th Digital Avionics Systems Conference,* October 2001.

[11] Jorge Aravena. Detecting change using pseudo power signatures. *Proc. 2002 IFAC Congress*, Barcelona, Spain, July 2002.

[12] T. Kourti. Process analysis and abnormal situation detection: from theory to practice. *IEEE Control Systems*, 22(5), pp. 10-25, 2002.

[13] J.J. Gertler. *Fault Detection and Diagnosis in Engineering Systems*. Marcel Dekker Inc., 1998.

[14] Michèle Basseville and Igor V. Nikiforov. *Detection of Abrupt Changes - Theory and Application*, Prentice-Hall, Inc, 1993.

[15] Venkat Venkatasubramanian, Raghunathan Rengaswamy. A review of process fault detection and diagnosis Part III: process history based methods, *Computers and Chemical Engineering* 27, pp. 327-346, 2003.

[16] J. Chen, R.J. Patton and H. Zhang. Design of unknown input observers and robust fault detection filters, *International Journal of Control*, vol. 63, no.1, pp. 85-105, 1995.

[17] N. Viswanadham and R. Srichander. Fault detection using unknown-input observers. *Control Theory and Advanced Technology* 3, pp.91-101, 1997.

[18] M. Basseville and I. Nikiforov. Fault isolation for diagnosis: nuisance rejection and multiple hypotheses testing, *15th IFAC World Congress, Barcelona*, IFAC, July 2002.

[19] R. Reiter. A theory of diagnosis from first principles. *Artificial Intelligence* 32 (1), pp. 57-95, 1987.

[20] W. Becraft and P. Lee. An integrated neural network/expert system approach for fault diagnosis. *Computers and Chemical Engineering* 17 (10), pp.1000-1014, 1993.

[21] E. Tarifa and N. Scenna. Fault diagnosis, directed graphs, and fuzzy logic. *Computers and Chemical Engineering* 21, pp.649-654, 1997.

[22] D. Chung and M. Modarras. A method of fault diagnosis: presentation of a deep-knowledge system. *AIChE National Meeting*, New York, NY, November, 1987.

[23] S. H. Rich and V. Venkatasubramanian. Model-based reasoning in diagnostic expert systems for chemical process plants. *Computers and Chemical Engineering* 11, pp.111-122, 1995.

[24] J. T. Cheung. Representation and extraction of trends from process data, *Doctor of Science Thesis*, Massachusetts Institute of Technology, Cambridge, MA, 1992.

[25] K. B. Konstantinov and T. Yoshida. Real-time qualitative analysis of the temporal shapes of the (bio) process variables. *American Institute of Chemical Engineers Journal* 38 (11), pp.1703-1715, 1995.

[26] J. Gertler. Intelligent supervisory control. *Artificial Intelligence Handbook, Volume II*. Research Triangle Park, NC, USA, 1999.

[27] S. Dash, R. Rengaswamy and V. Venkatasubramanian. A novel interval-halving algorithm for process trend identification. *The 4th IFAC Workshop on On-line Fault*

Detection and Supervision in the Chemical Process Industries, June 7- 8, Jejudo Island, Korea, pp.173-178, 2001.

[28] F. Ortega, J. B. Ordieres, C. Menendez and Nicieza C Gonzalez. Development of neural-based diagnostic system to control the ropes of mining shifts. *Proceedings of the International Conference in Ales, France*, pp.108-111, 1995.

[29] K. Watanabe, S. Hirota, D. M. Himmelblau. Diagnosis of multiple simultaneous fault via hierarchical artificial neural networks. *American Institute of Chemical Engineers Journal* 40 (5), pp.839-848, 1994.

[30] B. R. Bakshi and G. Stephanopoulos. Wave-net: a multiresolution, hierarchical neural network with localized learning. *American Institute of Chemical Engineers Journal* 39 (1), pp. 57-81, 1993.

[31] I.H.Witten and E.Frank. *Data Mining: Practical Machine Learning Tools and Technique with Java Implementations*, Morgan Kaufman Publishers, 2000.

[32] B.Scholkopf, R.C. Williamson, A.J. Smola, J. Shawe Taylor, J. Platt. Support vector method for novelty detection. *Neural Information Processing Systems*, pp 582-588, 2000.

[33] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and knowledge discovery*, Vol. 2. Number 2, 1998.

[34] T. Yairi, Y. Kato and K. Hori. Fault Detection by Mining Association Rules from House-keeping Data. *Proceedings of International Symposium on Artificial Intelligence, Robotics and Automation in Space* (ISAIRAS), 2001.

[35] W. Lee, S. J. Stolfo, K. Mok. Data mining in work flow environments: Experiences in intrusion detection. *Proceedings of Conference on Knowledge Discovery and Data Mining (KDD-99),* 1999.

[36] E. Eskin, Anomaly detection over noisy data using learned probability distributions. *Proceedings of the Seventeenth International Conference on Machine Learning,* 2000.

[37] Wenke Lee, Sal Stolfo, Kui Mok. Adaptive intrusion detection: a data mining approach, *Artificial Intelligence Review*, Kluwer Academic Publishers, 14(6): pp. 533-567, December 2000.

[38] M. Moya, M. Koch, and L. Hostetler. One-class classifier networks for target recognition applications. *Proceedings World Congress on Neural Networks*, pp.797–801, 1993.

[39] G. Ritter, and M. Gallegos. Outliers in statistical pattern recognition and an application to automatic chromosome classification. *Pattern Recognition Letters,* 18: pp. 525–539, 1997.

[40] L. Tarassenko, P. Hayton, M. Brady. Novelty detection for the identification of masses in mammograms. *Proc. of the Fourth International IEE Conference on Artificial Neural Networks*, volume 409, pp. 442–447, 1995.

[41] M. Moya, D. Hush. Network constraints and multiobjective optimization for one-class classification. *Neural Networks*, 9(3): pp.463–474, 1996.

[42] D.M.J. Tax. *One-Class Classification; Concept Learning in the Absence of Counter-Examples*, Ph.D. thesis Delft University of Technology, Delft, 2001.

[43] N. Japkowicz, C. Myers, M. Gluck. A novelty detection approach to classification. *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pp. 518–523, 1995.

[44] L. Tarassenko, P. Hayton, M. Brady. Novelty detection for the identification of masses in mammograms, *Proc. of the Fourth International IEE Conference on Artificial Neural Networks*, volume 409, pp. 442–447, 1995.

[45] A. Ypma, P. Pajunen. Rotating machine vibration analysis with second-order independent component analysis. *Proceedings of the First International Workshop on Independent Component Analysis and Signal Separation*, ICA'99, pp. 37–42, 1999.

[46] Dennis DeCoste. Learning envelopes for fault detection and state summarization. *IEEE Aerospace Conference*, March 2000.

[47] C. B. Barber, D. P. Dobkin, H. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Software* 22, pp. 469–483,1996.

[48] C.B Barber. *Computational Geometry with Imprecise Data and Arithmetic*. PhD thesis, Princeton University, 1992.

[49] C. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Walton Street, Oxford, 1995.

[50] Pallavi Chetan. *Blind Fault Detection Using Spectral Signature*, MS Thesis, Louisiana State University, May 2003.

[51] Venkata Vongala. *Knowledge-Based Fault Detection Using Time-Frequency Analysis*, MS Thesis, Louisiana State University, Aug 2005.

[52] V.N. Vapnik. *The Nature of Statistical Learning Theory*, Springer, 1995.

[53] V.N. Vapnik. *Statistical Learning Theory*, Wiley, 1998.

[54] N. Cristianini, J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000.

[55] E. Osuna, R. Freund, F. Girosi. Support vector machines: training and applications. *AI Memo 1602*, MIT, May 1997.

[56] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2): pp.1-47, 1998.

[57] Robert Burbidge. An introduction to support vector machines for data mining. *Young OR Conference,* The University of Nottingham, pp. 27 – 29 March 2001.

[58] F. Mörchen. Time series feature extraction for data mining using DWT and DFT. *Technical Report No. 33, Departement of Mathematics and Computer Science Philipps-University Marburg*, 2003.

[59] Charu C. Aggarwal, Alexander Hinneburg, Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional space. *Lecture Notes in Computer Science*, 1973:420, 2001.

[60] R. Agrawal, K. Lin, H. Sawhney, K. Shim. Fast similarity search in the presence of noise, scaling and translation in time-series databases. *Proceeding of the VLDB Conf.*, pp. 490-501, 1995.

[61] Yi-Leh Wu, Divyakant Agrawal, Amr El Abbadi. A comparison of DFT and DWT based similarity search in time-series databases. *CIKM*, pp. 488-495, 2000.

[62] J. Yoon, Y. Luo, J. Nam. A Bitmap approach to trend clustering and prediction in time-series databases. *Proc. of Data Mining and Knowledge Discovery: Theory, Tools, and Technology II*, Florida, April 2001.

[63] H.Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini. Text Classification using String Kernels. *Journal of Machine Learning Research*, (2), pp 419-444, 2002.

[64] S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, Vol IT-28, pp. 129-137, March, 1982.

[65] C. Leslie, E.Eskin and W.S. Nobel. The spectrum kernel: A string kernel for SVM protein classification. *Proceedings of the Pacific Biocomputing Symposium*, 2002.

[66] L.Botton, C. Cortes, J. Denker, H.Drucker, V.Vapnik. Comparison of classifier methods: a case study in handwriting digit recognition. *International Conference on Pattern Recognition*, pp 77-87, 1994.

[67] S.Knerr, L. Personnaz, G.Dreyfus. Single-layer learning revisited: A stepwise procedure for building and training a neural network,. *Neurocomputing: Algorithms, Architectures and Applications,* 1990.

[68] Chih-Wei Hsu, Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, Vol. 13, No. 2, pp. 415 – 425, March 2002.

[69] J. Friedman. *Another Approach to Polychotomous Classification*, Technical Report, Department of Statistics, Stanford University, 1996.

[70] J. Weston, C. Watkins. *Multi-class support vector machines*. Technical Report CSD-TR-98-04, Department of Computer Science, Royal Holloway, University of London, Egham, TW20 0EX, UK, 1998.

[71] Van Der Linden. C.A.A.M., DASMAT - *Delft University Aircraft Simulation Model and Analysis Tool, A Matlab/Simulink Environment for Flight Dynamics and Control Analysis, Report LR-781*, Delft University of Technology, Delft, 1996.

[72] J.C.Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods - Support Vector Learning,* Cambridge MA: MIT Press**,** pp. 185-208, 1998.

Appendix: User Manual

I. One class classification

1. Create a directory where to put the training and test data.
2. Run main.m (under the 'Convex' folder) to perform one-class classification

(1) Load the normal data from the directory where test data is stored.

(2) Choose how many state variables you want to use to build the normal convex hull.

n = input('Enter the index of state variables used for building normal convex hull: 1 -- Roll Rate; 2 -- Pitch Rate; 3 -- Yaw Rate; 4 -- True Airspeed; 5 -- Angle of Attack; 6 -- Slideslip Angle; 7 -- Roll Angle; 8 -- Pitch Angle; 9 -- Yaw Angle;10 -- Altitude ');

(3) Load the test data

II. Two class classification

Installation:

Copy the "SVM_Software" folder into the MATLAB folder containing all of the program files for MATLAB. Then start MATLAB and open the Path Browser under File→Set Path (see Figure 1 below). Click on "Add with Subfolders", and then browse for the SVM Software folder. Click once on the folder icon next to "SVM_Software", and then click OK. The new folders should now be added to the top of the MATLAB search path list and highlighted. Click the Save button in the lower left corner of the Path Browser to save these folders to the MATLAB search path and close out of the Path Browser by clicking the Close button.

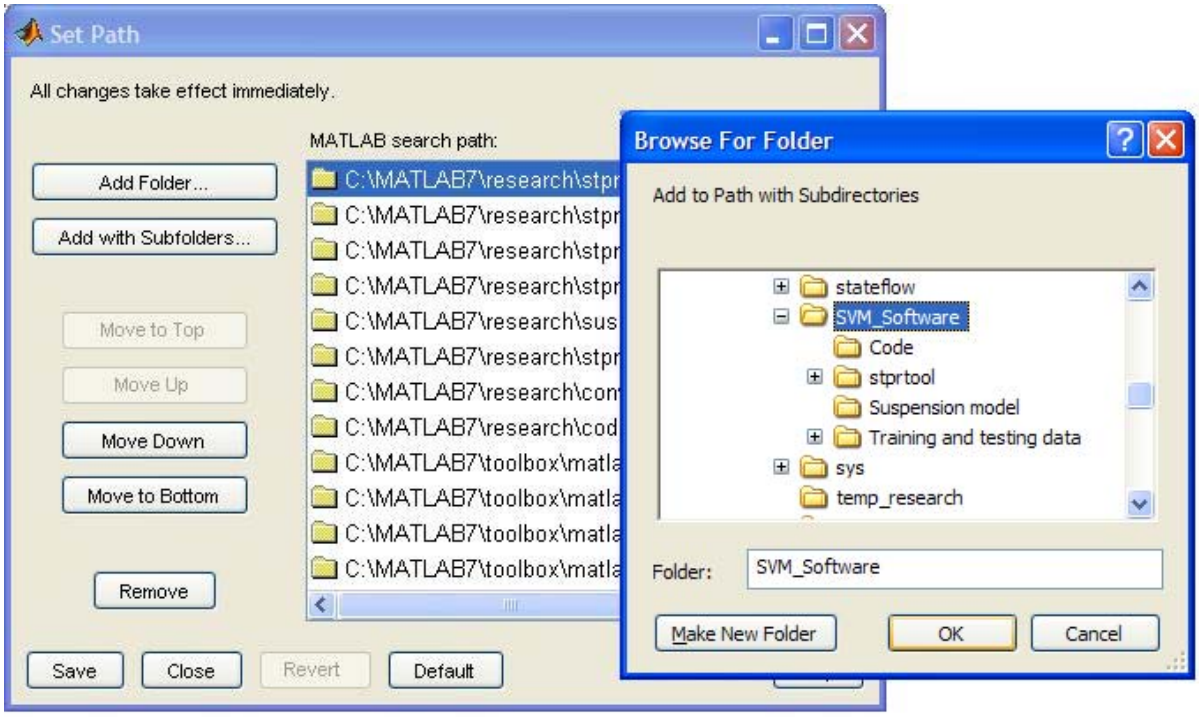


Figure 1

Purpose:

The binary SVM classifier software is designed to create a binary SVM classifier using system data from two distinct categories-typically normal and faulty-and then classify unknown system data into one of these two categories.

Training Data Requirements:

The training data used to create the binary SVM classifier may be created using SIMULINK or some other modeling software, but must be formatted so that the normal and faulty data are data fields in a MATLAB structure which includes a data field containing the time scale for the normal and faulty data. Each simulation or experiment for the training data should be stored as a column vector of type double (or any other floating point precision), and there should be an equal number of simulations for normal

and faulty data. The total number of simulations created in the structure is partitioned between the training data and the SVM model testing data.

SVM Model Testing Data:

The SVM model testing data is used to test the trained SVM classifier for false alarm percentage, missed alarm percentage, and time delay and is taken from the same data set as the training data. That is, the SVM model testing data is stored in the same MATLAB structure as the training data, in the same data fields for normal, faulty and time data. So, if there is a total of n simulations, and m of these are chosen to be used for training then there are (n-m) experiments available for testing the SVM model.

Creating Training and Testing Kernels:

To create the string based training and testing kernels, run the m-file main_kernel.m. A window will pop up prompting the user to "Find the file which contains the training data." This file is the MATLAB binary file containing the training data saved in a MATLAB structure as described in Training Data Requirements. MATLAB then prompts the user to enter the name of the structure containing the training data. The names of the variables in the file will be displayed so the user will be able to type the name of the structure. Once the structure name is entered, the field names print to the command window, and MATLAB prompts for the name of the data field which contains the normal data. Type the name and press enter. The same prompt occurs for the faulty data and the time data. Then, MATLAB prompts for the number of experiments to use for training. Using more experiments will take longer to train, but will be more accurate in classifying unknown data. If the number of data points for each simulation is between 1000 and 5000, then 2 to 5 training experiments will run in a reasonable amount of time.

Next, MATLAB prompts for the number of experiments to use for testing. This is the number of experiments which will be used in testing the trained SVM classifier (see Testing the Trained SVM). As the script file runs, it will plot all of the experiments used for training and testing using the time data given by the user. Also, histograms will be plotted for the normal and faulty data. This script file also creates the non-uniform partitioned region set with its corresponding symbol set, the subsequence library, and the training feature vectors. At the end, it prints the training kernel in the form of a character matrix to the command window with commas between matrix elements. Finally, the user is prompted to choose locations to save the data for training, data to use for testing the SVM, the training kernel, the testing kernel, and the structure for testing new data sets. The folder, Training and testing data is where these files should be saved. There are subfolders: train data, test data, training kernels, test kernels, and test structures for saving each of the files, respectively. The training data in folder train data is used for training the binary SVM classifier (see Training the Binary SVM Classifier), the SVM model testing data in test_data and the testing kernel in Test kernels are used to test the trained binary SVM classifier (see Testing Trained Binary SVM Classifier), the test structure saved in test structures is used to test unknown data sets (see Testing Unknown Data Sets), and the training kernel saved in Training kernels is the character matrix which printed to the command window. It is saved in case the user accidentally clears the screen before performing the necessary steps described next in Training the Binary SVM Classifier.

Training the Binary SVM Classifier:

Once main_kernel.m is completed, the training kernel matrix will print to the command window. Copy the entire matrix. Then open kernel_fun.c in WordPad (go to the kernels folder: stprtools→kernels). At the bottom of this file, within the function, kernel, there is a two dimensional matrix, 'mat'. Delete the contents between the braces and paste the training kernel matrix. Look at the dimensions of the training kernel matrix, Ktrn, in the MATLAB workspace, and change the dimensions to which mat is initialized so that mat has the same dimensions as Ktrn. Save kernel_fun.c and close the file.

Next, change the current directory to the stprtool folder. Then type "stprpath" at the command prompt in the command window and press enter. This adds a path for the SVM toolbox. Next, type "compilemex" at the command prompt to compile all of the mex-files used for training the binary SVM classifier. The first time this command is run, MATLAB will prompt for the user to choose a compiler. Choose the compiler with MATLAB in its directory path.

Once the mex-files are compiled, the binary SVM classifier can be trained by running the m-file, test_SVM.m. Whenever this script file runs, the user is prompted to "Find the file with the trained data." This is the training data which was saved in the subfolder train data under Training and testing data at the end of main_kernel.m. Go to this subfolder and click once on the name of the file which was saved at the end of main_kernel.m and press open. At the end of this script file, the SVM classifier will be trained and it will be stored in a MATLAB structure named model. The user will be prompted to "Choose a name and location to save the trained SVM model." This is the binary SVM classifier. There is a subfolder of Training and testing data named SVM

models in which to save the binary SVM classifier. It is recommended that the user choose a very descriptive name.

Testing the Trained SVM:

To test the trained SVM model, the function svmclass can be used. It takes one input: the trained SVM model (which is in the MATLAB workspace after buildSVMmodel.m is completed). If the trained SVM model is not currently in the workspace, the user can load it by clicking File→Open→Training and testing data→SVM models, and opening the desired trained SVM model. There is only one output from this function: a structure which contains (among other things) the percentage of false alarms, missed alarms, and the time delay of the trained SVM. Whenever this function is run, the user is prompted to "Open the data file with the original test data." This "original test data" is the SVM model testing data which was saved at the end of main_kernel.m in the subfolder, test_data. Go to the test_data subfolder, click on the name of the file and click open. After this file is loaded, the user is prompted to "Open the data file with the test kernel, Ktst". This too was saved at the end of main_kernel.m in the subfolder of Training and testing data named Test kernels. When the function finishes the user can examine the results.

Quick Start: B747 Model

1. Creating Training and Testing Kernels

Type "main_kernel" at the command prompt in the command window and press enter.

(1) "Enter the number of state variable: "

You need to run it for each of seven state variables. Therefore, you have to specify the index of state variable.

4 -- 'True Airspeed'; 5 -- 'Angle of Attack'; 6 -- 'Slideslip Angle'; 7 --'Roll Angle'; 8 -- 'Pitch Angle'; 9 -- 'Yaw Angle'; 10 --'Altitude'

(2) Enter the number of training data simulations:

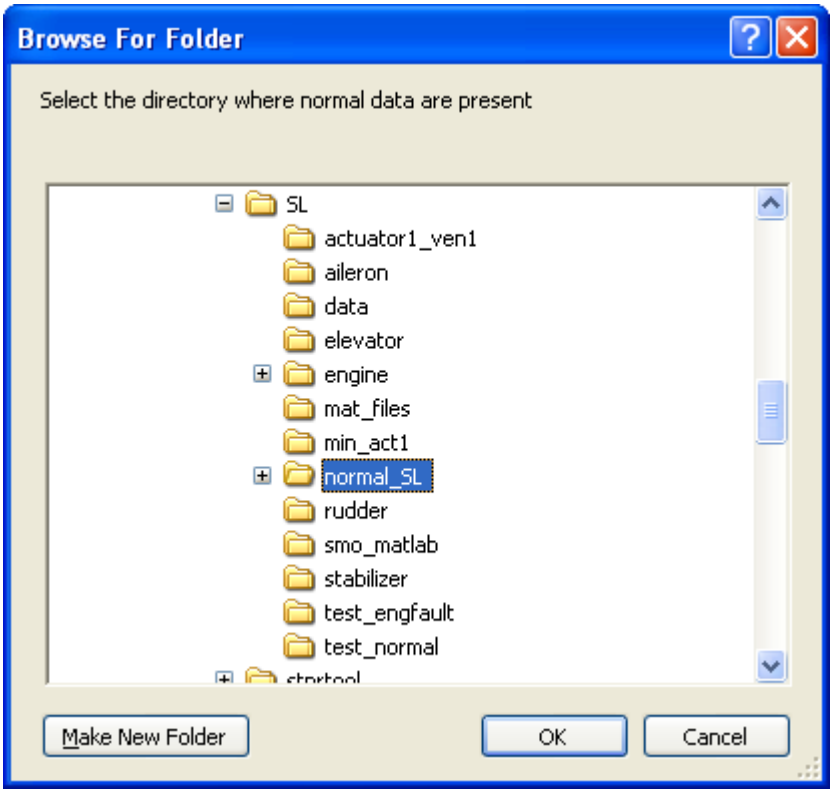
You enter how many training set you want for normal/faulty data. Each simulation contains 3000 data points.

(3) Enter the number of test data simulations:

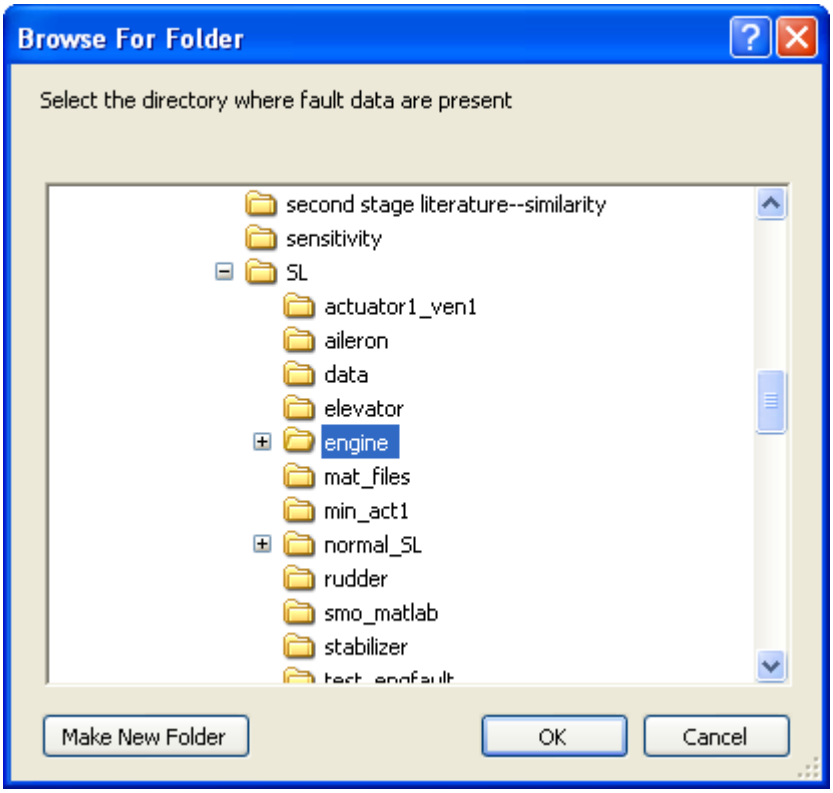
(4) Enter the type of fault: 1--elevator, 2--aileron, 3 --rudder, 4--stabilizer, 5--engine

You need to pick the type of the fault that you will perform the detection for.

(5) Enter the directory where the normal data stays:



(6) Enter the directory where the faulty data stays:



By selecting a particular fault, the program loads all the files corresponding to that particular fault.

2. Test trained SVM

Open the 'Ktrn' saved by main_kernel.m, make a copy of all data set. Then open the kernel_fun.c (under the 'stprtool/kernels' folder) paste the Ktrn to the matrix 'mat' in the function "double kernel (long a, long b)"

(1) Compile the mex file. Change matlab path into the directory of mex file. Type 'stprpath', then followed by 'compilemex'.

(2) Run test_SVM.m for every state variable, then run voting.m for final fault detection decision with false alarm and missed alarm rate and time delay.

3. Fault Isolation

(1) Compile the mex file. Change matlab path into the directory of mex file. Type 'stprpath', then followed by 'compilemex'.

(2) Run main_isolation.m

[1] "Enter the number of state variable: "

You also need to run it for each of seven state variables. Therefore, you have to specify the index of state variable. 4 -- 'True Airspeed'; 5 -- 'Angle of Attack'; 6 -- 'Slideslip Angle'; 7 --'Roll Angle'; 8 -- 'Pitch Angle'; 9 -- 'Yaw Angle'; 10 --'Altitude'

[2] Enter the number of training data simulations:

[3] Enter the number of test data simulations:

Run fault_isol.m for each of seven states.

Matlab Code:

The Matlab codes for one-class classification and two-class classification are available upon request. Please contact Dr. Jorge Aravena in ECE department at Louisiana State University. His email address is aravena@ece.lsu.edu.

Vita

Min Luo is currently a doctoral student in the Electrical and Computer Engineering Department at Louisiana State University. She received a Bachelor of Science degree in electrical engineering from Taiyuan Technological University in July 1996. She entered the graduate program in the Department of Electrical and Computer Engineering at Louisiana State University in the spring of 2002, and will receive the degree of Doctor of Philosophy in December 2006.